J.P. LAMOITIER

```
0010 LET A$="LE LANGAGE "
0020 LET B$="BASIC"
0030 PRINT A$,B$,"  ",A$,B$
0040 PRINT A$,B$,"  ",A$,B$
0050 PRINT
0060 PRINT TAB(12),"ET SES EXTENSIØNS"
0070 END
```



Eyrolles
EDITEUR-PARIS

LE LANGAGE BASIC
ET SES EXTENSIONS

R4e

OUVRAGES DU MÊME AUTEUR

Le langage FORTRAN IV, par J.-P. LAMOITIER (Dunod) (1978).

Exercices de programmation en Fortran IV, par J.-P. LAMOITIER (Dunod).

LE LANGAGE BASIC
ET SES EXTENSIONS

par

Jean-Pierre LAMOITIER

Ingénieur Arts et Métiers
Docteur 3e cycle Mathématiques Appliquées I.C.G.
Ingénieur Conseil en Informatique

TROISIÈME ÉDITION

Nouveau tirage

ÉDITIONS EYROLLES
61, boulevard Saint-Germain – 75005 PARIS
1980

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service « Presse », Éditions EYROLLES,
61, Boulevard Saint-Germain, 75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement et sans aucun engagement de votre part, un avis de parution des nouveautés en vente chez votre libraire habituel.

TABLE DES MATIÈRES

TABLE DES EXERCICES

Sujet	*Enoncé*	*Organigramme*	*Programme*
Table de Pythagore	12.1.3	12.2.3	12.2.3
PGCD de deux nombres	1.3.1.	1.3.1	3.7
Recherche de nombres premiers	1.6.2	1.6.3	6.3
Recherche des nombres parfaits	12.1.2	12.2.2	12.2.2
Équation du second degré	12.1.5	12.2.5	12.2.5
Calcul de π par des séries	2.5	2.5	4.8
Calcul de e par la série $\frac{1}{n!}$	1.6.1	1.6.3	1.4.5 et 3.7
Calcul du nombre d'or	12.1.1	12.2.1	12.2.1
Résolution d'une équation par la méthode de Newton	1.1.3	1.2	
Calcul de la racine n eme d'un nombre par itérations	1.3.2	1.3.2	
Tabulation de fonctions	5.6.1		5.6.2
Recherche du plus grand élément d'un tableau	3.7	3.7	3.7
Calcul de $n!$	6.3.1	6.3.2	6.3.2
Tracé de courbe	5.6.1	5.6.2	5.6.2
Régression linéaire simple	7.6.2	7.6.2.1	7.6.2.1
Régression linéaire multiple	7.6.2.2	7.6.2.2	7.6.2.2
Gestion de stock	10.6	10.6.2	10.6.4
Remboursement d'un capital	12.1.6	12.2.6	12.2.6
Rentabilité d'un investissement	12.1.7	12.2.7	12.2.7
Volume d'une cuve cylindrique	12.1.4	12.2.4	12.2.4
Résolution d'un système linéaire	12.1.8	12.2.8	12.2.8
Minimisation de câblage (algorithme de Kruskal)	12.1.9	12.2.9	12.2.9

AVANT-PROPOS

Le Langage BASIC a été initialement conçu pour familiariser rapidement les débutants à l'Informatique, notamment à partir de terminaux connectés à un système « temps partage ». Facile à apprendre, il s'est rapidement développé en même temps que les systèmes « temps partagé ».

Après un temps d'arrêt, BASIC connaît une nouvelle jeunesse avec le développement des ordinateurs de tables à vocation scientifique et instrumentale et plus récemment avec l'apparition des micro-ordinateurs à usage personnel (les « home computers »).

Ce succès est dû à la simplicité du langage : l'introduction des données est facile grâce au format libre, les boucles de calcul ont une forme proche du langage naturel, la manipulation des tableaux est aisée et tout ceci avec un nombre restreint d'instructions dont la syntaxe est élémentaire.

Très dépouillé à l'origine, il s'est progressivement enrichi de nombreuses extensions qui le rendent apte à traiter des problèmes relativement complexes. Elles ont porté d'abord sur les possibilités algorithmiques puis sur l'utilisation de petits ordinateurs en instrumentation ; ensuite ont été ajoutées des possibilités graphiques (tracé de courbes). Enfin des extensions ont été apportées pour les « home computers » qui font appel à des périphériques bon marché.

Le BASIC constitue pour de nombreuses personnes (étudiants, médecins, gestionnaires, etc.) une première approche de l'ordinateur. Cet ouvrage a donc été conçu en vue de leur faciliter cette approche. Pour cela un mode d'exposition assez didactique a été adopté. L'exposé du BASIC se fait progressivement au fil des différents chapitres dont chaque introduction présente le contenu. Les différents paragraphes de cours contiennent de nombreux exemples et des exercices accompagnés de leur solution.

Afin de permettre au lecteur d'assimiler aisément le langage BASIC et de lui faciliter d'autres ouvertures, le premier chapitre a été consacré

à des généralités relatives aux langages de programmation. De même un exposé général sur les fichiers précède la description des instructions d'entrée et de sortie relatives aux fichiers, de nombreuses remarques jalonnant l'exposé.

Le BASIC a subi de multiples extensions souvent incompatibles d'un constructeur à l'autre. Seules les principales ont été regroupées dans le chapitre 11, afin de ne pas alourdir inutilement les premiers chapitres.

Le dernier chapitre a été consacré à des exercices de récapitulation qui ont été choisis en tenant compte de leur caractère pédagogique et de leur intérêt sur le plan des applications. Certains exercices plus ardus sont destinés au lecteur soucieux de se perfectionner en vue d'aborder ensuite des problèmes plus difficiles.

Pour faciliter la lecture deux jeux de caractères d'imprimerie ont été utilisés : le lecteur pourra dans une première lecture « faire l'impasse » sur les paragraphes écrits en petits caractères et y revenir par la suite.

L'expérience montre par ailleurs que les débutants éprouvent des difficultés à dessiner les organigrammes. Pour cette raison, chaque exercice comporte une solution composée d'explications préliminaires, d'un organigramme et ensuite du programme BASIC correspondant.

Enfin, pour rendre ce livre plus commode, une table permet au lecteur de trouver rapidement les exercices qui l'intéressent particulièrement en vue d'une application déterminée.

Nous espérons que ce livre permettra au lecteur d'acquérir rapidement les connaissances nécessaires à l'utilisation d'un ordinateur en particulier via un terminal de Time-sharing et lui donnera les ouvertures permettant d'accéder à des langages plus compliqués ou plus spécialisés et de traiter ainsi des applications diverses.

CHAPITRE 1

INTRODUCTION A LA PROGRAMMATION ET AUX LANGAGES DE PROGRAMMATION

1.1. Notion d'algorithme.

L'évolution des mathématiques et le développement des « mathématiques appliquées » ont contribué à donner une importance à la notion d'ALGORITHME, qui apparaît après les notions de théorème d'existence, théorème de calcul ainsi que nous allons le voir.

1.1.1. Théorème d'existence.

Un théorème d'existence affirme l'existence d'une solution à un problème donné et les conditions nécessaires à cette existence. Par exemple pour la résolution de l'équation $f(x) = 0$, nous pouvons citer le théorème d'existence suivant :

x étant une variable réelle, l'équation $f(x) = 0$ admet une racine x_0 située dans l'intervalle (a, b) si les conditions suivantes sont satisfaites :

— $f(a)$ et $f(b)$ sont de signes contraires,
— $f(x)$ est définie et continue dans cet intervalle.

1.1.2. Théorème de calcul.

Les théorèmes d'existence ne donnent aucune indication sur la méthode permettant de trouver la solution. D'autres théorèmes indiquent des méthodes de calcul permettant d'atteindre effectivement la solution cherchée. Reprenons l'exemple précédent. Il existe une méthode dite de Newton (ou encore de la tangente dans la version simplifiée) permettant d'attendre la racine de $f(x) = 0$.

Si x_0 est une valeur approchée de la racine et, si $f(x)$ est continue et dérivable, en calculant les valeurs x_i données par :

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}.$$

On obtient une suite $x_0, x_1, \ldots x_i$ dont les valeurs convergent vers une racine de $f(x) = 0$.

(Cet énoncé n'est valable que sous certaines conditions).

1.1.3. Algorithme.

Le théorème précédent est insuffisant pour mener à bien le calcul : il ne précise pas comment obtenir x_0 et ne donne aucun renseignement sur le nombre d'itérations à effectuer pour obtenir une valeur approchée très convenable de la racine. *L'algorithme* doit décrire avec précision et sans aucune ambiguité le processus de calcul permettant d'obtenir le résultat. Ici nous pouvons utiliser l'algorithme suivant :

1. données si x_0 valeur de départ
   ε précision absolue recherchée sur $f(x)$
   n nombre maximum d'itérations,
2. Calcul $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$,
3. si $|f(x_1)| < \varepsilon$ arrêter le calcul $\varepsilon = 0{,}001$ par exemple,
4. sinon remplacer la valeur de x_0 par celle de x_1

$$x_1 \rightarrow x_0$$

incrémenter de 1 le nombre i d'itérations
si $i > n$ imprimer « pas de convergence »,
si $i \leqslant n$ aller en 2.

1.2. Notion d'organigramme (1).

Pour décrire un procédé de calcul ou plus généralement de traitement il est préférable de tracer un *organigramme*. Il s'agit d'un schéma composé essentiellement de

— rectangles,
— losanges,
— segments de droite avec flèches,
— parallélogrammes.

(1) On dit aussi un ordinogramme.

Un rectangle contient une opération de traitement plus ou moins compliqué à réaliser. Un losange représente une opération de test d'où l'on sort par deux directions au moins selon le résultat du test. Un parallélogramme indique une opération d'entrées/sorties, c'est-à-dire d'échange d'information entre l'être humain et l'ordinateur, ou entre la mémoire centrale et une mémoire auxiliaire.

Exemple : Résolution de l'équation du second degré $ax^2 + bx + c = 0$.
Calcul de $\Delta = b^2 - 4ac$.
Si $\Delta > 0$ deux racines réelles distinctes.
Si $\Delta < 0$ pas de racines réelles.
Si $\Delta = 0$ une racine double.

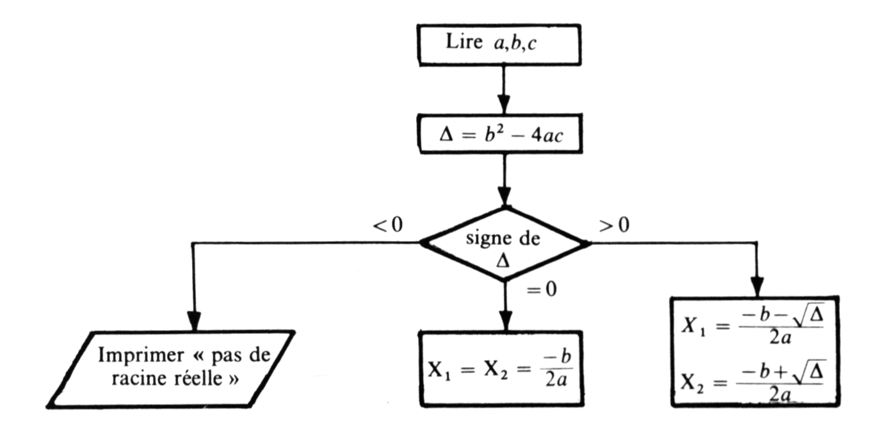


FIG. 1.1.

La méthode exposée rapidement s'applique aisément à la méthode de Newton (fig. 1.2) :

Dès que le procédé de calcul devient un peu compliqué la méthode de l'organigramme devient supérieure à celle indiquée en 1.1.3. Très souvent un organigramme ne peut tenir sur une page. Il a donc fallu prévoir des symboles spéciaux pour les renvois. En outre afin de faciliter la lecture des organigrammes par différentes personnes, l'AFNOR a fait paraître une norme (1) qui précise les différents symboles à utiliser.

(1) Norme AFNOR... REF Z67 010.

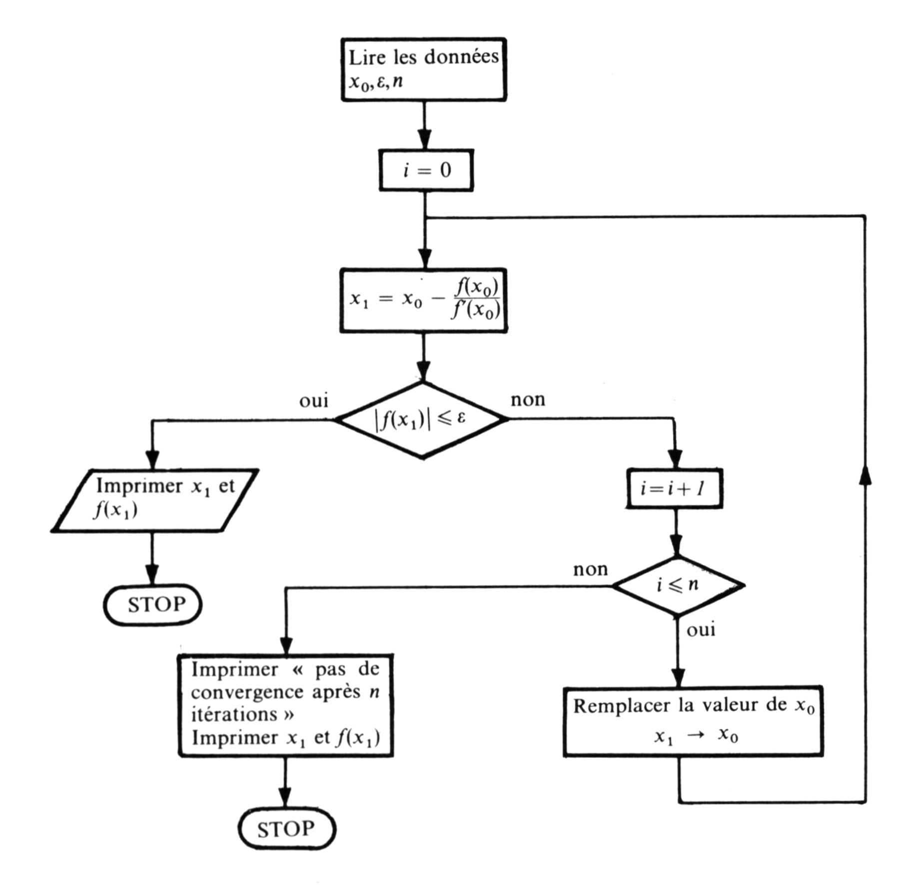


FIG. 1.2.

1.3. Exemples de tracés d'organigrammes.

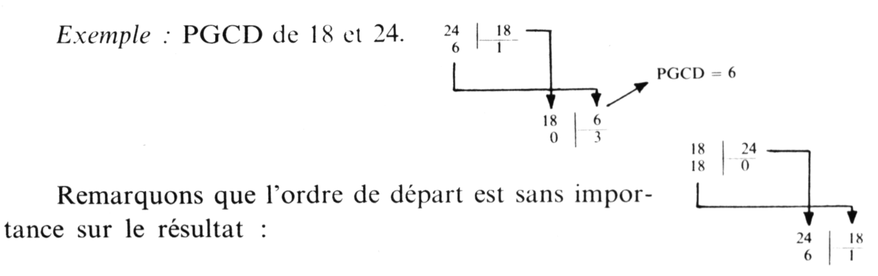
Les exemples suivants très simples seront repris lors de la deuxième partie, afin de constituer des programmes BASIC simples.

1.3.1. Calcul du PGCD de deux nombres.

Tracer l'organigramme permettant de déterminer le PGCD de deux nombres, à partir de la méthode suivante :

Le Plus Grand Commun Diviseur de deux nombres A et B peut s'obtenir par divisions successives de A par B, puis de B par le reste obtenu et ainsi de suite, jusqu'à ce que l'on obtienne un reste nul, le dernier diviseur est alors le PGCD de A et B.

Exemple : PGCD de 18 et 24.



Remarquons que l'ordre de départ est sans importance sur le résultat :

Pour tracer aisément cet organigramme, nous utiliserons le symbole $\div$ qui caractérise la division entière :

$$15 \div 2 \text{ donne pour quotient } 7.$$

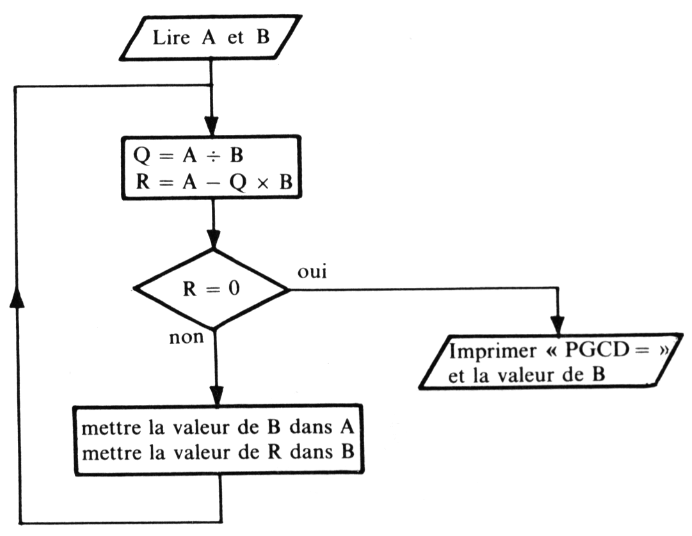


FIG. 1.3.

1.3.2. Calcul de la racine n^eme d'un nombre par itérations.

On démontre aisément par l'intermédiaire de la méthode de Newton que la suite

$$x_{i+i} = \frac{A}{n x_i^{n-1}} + \frac{n-1}{n} x_i$$

converge sous certaines conditions vers $\sqrt[n]{A}$.

Écrire l'organigramme permettant d'utiliser cette méthode. Les données sont A, n, m et la précision demandée est ε. On arrêtera les itérations lorsque l'une des conditions suivantes sera satisfaite :

$$|A - x_i^n| \leqslant \varepsilon$$

ou

$$\text{nombre d'itérations effectuées} = m$$

on prendra pour valeur de départ $x_0 = 2$.

Solution :

Cette méthode fait apparaître qu'il faut connaître x_{i-1} pour calculer x_i, et que les éléments x_{i-2}, x_{i-3}, etc., de la suite, sont sans intérêt.

Il suffit donc de « mémoriser » le dernier résultat calculé pour pouvoir continuer les itérations.

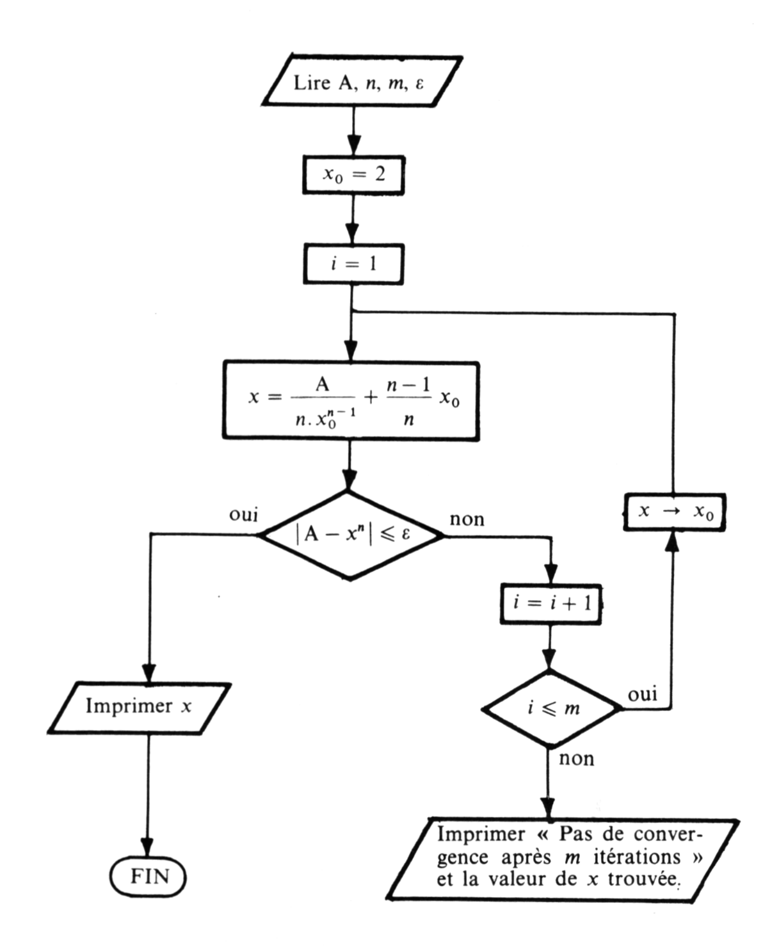


FIG. 1.4.

1.4. Généralités sur les langages de programmation.

1.4.0. Introduction.

L'emploi des premiers ordinateurs électroniques exigeait de la part de l'utilisateur un apprentissage du « code interne » de la machine appelé plus généralement « langage machine ». Ce mode de programmation est très pénible : l'écriture, la mise au point d'un programme sont longues. Afin de dispenser l'utilisateur d'un tel effort, les constructeurs et les universités ont développé, d'une part des « langages d'assemblage » qui apportent un certain allègement pour le programmeur, d'autre part des langages plus synthétiques que l'on appelle maintenant « langage de programmation ». Ils sont composés de mots du langage humain et de symboles mathématiques plus faciles à apprendre et à utiliser que le « langage machine ».

Alors que le « langage machine » diffère d'une machine à l'autre, les « langages de programmation » sont assez universels et ne nécessitent qu'une très courte adaptation pour passer d'un ordinateur à l'autre. Le mode de travail est le suivant :

— Écriture d'un programme en langage de programmation.

— Utilisation d'un programme appelé COMPILATEUR. Ce programme, fourni par le constructeur, traduit le programme écrit en langage évolué en programme exécutable par l'ordinateur (langage machine).

— Exécution du programme.

L'un des premiers langages de programmation fut FORTRAN (FORmula TRANslation) élaborée grace à l'initiative d'IBM et qui fut repris et développé par tous les autres constructeurs.

Ce langage très commode pour le calcul scientifique nécessite un certain temps d'apprentissage qui peut le rendre rébarbatif pour des utilisateurs occasionnels. Pour cette raison essentielle le Dartmouth College créa le langage BASIC.

De nombreux langages de programmation existent maintenant et nous pouvons les classer en différentes catégories.

Langages à usage général mais orientés néanmoins vers un type d'applications.

Citons par exemple :

— Le FORTRAN IV : calcul scientifique et technique. Ce langage permet une bonne efficacité à l'exécution. Il est utilisé parfois pour certaines applications de gestion (IBM 1130, PHILIPS P880) avec succès).

— L'ALGOL 60 (ALGOritmic Language) : D'une meilleure présentation, ce langage est surtout utilisé par les mathématiciens en particulier pour la publication des algorithmes. Ce langage a donné lieu a des extensions (Algol W et Algol 68).

— Le COBOL (Common Business Oriented Language) : Ce langage est utilisé presque exclusivement pour les applications de gestion. Très mal commode pour le calcul, il permet de traiter aisément des fichiers de données, ce qui est caractéristique des problèmes de gestion.

— Le PL/1 (Programming Language number 1) : Orienté à la fois vers les applications de gestion et de calcul scientifique, ce langage est très complet mais son assimilation est assez longue.

— Le BASIC : Langage d'apprentissage particulièrement rapide qui permet notamment l'utilisation d'un ordinateur à partir d'un terminal de « time sharing » de façon très commode.

Langages à usage particulier : ces langages sont très nombreux mais d'autant moins connus que leur domaine d'utilisation est limité. Citons par exemple :

— les langages de simulation : GPSS, SIMSCIPT, SIMULA, etc.,

— les langages destinés à la commande numérique de machines-outils : ADAPT, APT, IFAPT, AUTOSPOT, MINIAPT, etc.,

— les langages de contrôle automatiques,

— les langages destinés au traitement non numérique de l'information : LISP, SNOBOL, COMIT, FORMAC, etc.

1.4.1. Les différentes catégories d'instructions.

En reprenant l'un des organigrammes précédents, nous constatons qu'il existe :

— des instructions de calcul,

— des instructions de test (correspondant aux losanges),

— des instructions dites « d'entrées/sorties » correspondant à la lecture de données et à l'impression de résultats,

— des instructions de branchement « inconditionnel » permettant de sauter d'une instruction à une instruction autre que celle qui suit immédiatement dans le programme.

Nous constaterons cependant qu'il existe d'autres types d'instructions.

1.4.2. Notion d'étiquette.

Pour exécuter un branchement d'une instruction I_0 vers une instruction I_1, il faut pouvoir repérer l'instruction I_1. Pour cela l'utilisateur doit

pouvoir placer devant I_1 un repère auquel il peut se référer pour effectuer le branchement.

Ce repère est appelé ETIQUETTE (on dit aussi LABEL). Nous verrons plus loin que chaque instruction BASIC contient un repère très simple : un numéro de ligne.

1.4.3. Notion d'affectation : Instruction d'affectation.

En mathématiques, l'expression :

$$y = ax + b$$

signifie que y est une fonction dont la valeur formelle est $ax + b$.

En BASIC le fait d'écrire :

LET Y = $A * X$ + B

Y = AX + B ?
oui
non

Fig. 1.5.

signifie qu'il faut calculer l'expression à droite du signe = à partir des valeurs numériques de A, B, X et affecter le résultat à Y. Sous cette forme, le signe « = » est un symbole d'affectation.

Au contraire dans une instruction de test, par exemple dans la figure 1.5, le signe = de l'expression Y = A * X + B signifie qu'il faut calculer la valeur numérique de l'expression située à droite du signe = et comparer avec la valeur numérique de Y.

Dans l'écriture des organigrammes on pourra utiliser le signe = aussi bien pour l'affectation que pour la comparaison. Ceci est dû au fait que l'organigramme est lu par l'homme et non pas par la machine. Il peut donc faire la distinction aisément. Pour cette raison on pourra de même utiliser les lettres majuscules et minuscules, l'alphabet latin et l'alphabet grec, etc.

De même en BASIC, le signe = sera utilisé indifféremment pour les instructions d'affectation (1) et pour les tests

Remarques : Les valeurs numériques des variables sont « rangées » dans des « casiers » (emplacements de mémoire). Chaque variable a son casier. L'instruction LET Y = X revient à lire la valeur numérique située dans le casier X et à l'écrire dans le casier Y en remplacement de l'ancienne valeur numérique de Y.

(1) En ALGOL, la distinction se fait de la façon suivante :

- le symbole := est utilisé pour l'affectation,
- le symbole = est utilisé pour les comparaisons.

	X		Y	
	3,125		5,2	

Exemple : Contenu des casiers avant exécution

	X		Y	
	3,125		3,125	

après exécution de l'instruction :

LET Y = X

FIG. 1.6.

Par conséquent, une instruction d'affectation pourra être écrite sous la forme :

LET X = X + SIN (Y)

1.4.4. Les expressions arithmétiques.

Ces expressions destinées à représenter un calcul mathématique simple sont constituées à partir des éléments suivants :
— variables réelles indicées ou non,
— constantes réelles,
— fonctions standard (sinus exponentielle, etc.) et non standard,
— opérateurs arithmétiques,
— parenthèses gauche et droite.

Exemple :

A * B(I) + 3* COS(X) − FND(Y) ↑ 2/X

A et B (I)	sont des variables réelles
2 et 3	sont des constantes réelles
CØS	est un appel de la fonction standard cosinus
FND	est un appel à une fonction non standard
−, +, *, /, ↑	sont des opérateurs arithmétiques

Cette expression s'écrirait en mathématiques classiques

$$ab_i + 3 \cos x - \frac{(d(y))^2}{x} \qquad d(y) \text{ étant une fonction.}$$

L'écriture des expressions arithmétiques doit satisfaire à différentes règles que nous retrouvons dans de nombreux langages de programmation :

— Le jeu de caractères disponible est souvent réduit aux majuscules de l'alphabet latin, aux chiffres et à quelques caractères spéciaux. L'alphabet grec n'est pas disponible.

— Ces expressions ne peuvent s'écrire que de façon linéaire : $\frac{a}{b}$ doit s'écrire A/B.

— Les indices doivent être représentés soit entre parenthèses (BASIC, FORTRAN), soit entre crochets (ALGOL).

Exemple : $\frac{x_i}{n}$ doit s'écrire X (I)/N.

— Le nombre de parenthèses gauches doit être égal au nombre de parenthèses droites.

1.4.4.1. Évaluation d'une expression arithmétique.

Lorsqu'il faut effectivement calculer la valeur numérique d'une expression arithmétique, il importe de savoir dans quel ordre il faut effectuer les diverses opérations. Cet ordre apparaît évident pour l'être humain doté de « bon sens » mais ne l'est pas pour l'ordinateur. Il faut donc établir un algorithme permettant de déterminer dans quel ordre il faut « évaluer » une expression arithmétique.

Différentes méthodes ont été étudiées. La méthode la plus courante consiste à attribuer une « PRIORITE » aux différents opérateurs du langage. Nous reverrons ce problème en 1.4.9 à propos des compilateurs.

1.4.5. Exemple de programme BASIC.

L'exemple suivant correspond au calcul du nombre e par la série $\frac{1}{n!}$.

```
0010 LET E=F=N=1
0020 LET F=F/N
0030 LET X=E+F
0040 IF X=E THEN 80
0050 LET E=X
0060 LET N=N+1
0070 GØ TØ 20
0080 PRINT "NØMBRE D'ITERATIØNS =";N
0090 PRINT "E=";E
0100 STØP
0110 END
```

Les instructions d'étiquette :
— 10, 20, 30, 50, 60 sont des instructions d'affectation ;
— 40 est une instruction de branchement conditionnel ;
— 70 est une instruction de branchement inconditionnel
— 80 et 90 sont des instructions d'impression.

1.4.6. Notion de sous-programme.

Le mathématicien définit souvent une fonction nouvelle, par exemple :

$$f(x) = ax + b \quad \text{si} \quad x \geqslant 0$$
$$f(x) = 0 \quad \text{si} \quad x < 0$$

ou encore :

$$\Gamma(x) = \int_0^{\infty} f(x)e^{-x}dx.$$

Puis il continue à faire des calculs en utilisant cette fonction comme une simple opération ou plus exactement comme une fonction « standard » du type sinus ou logarithme

Ceci peut se réaliser en programmation, grâce à la notion de sous-PROGRAMME. En BASIC, cette notion a été peu développée, de façon à ne pas compliquer la structure du langage.

1.4.7. Notion de constante et de variable.

Dans l'expression : $y = ax + b + 3$ le mathématicien considère que :

3 est une constante,
a et b sont des paramètres,
x est une variable,
y est une fonction

Dans les langages tels que FORTRAN, ALGOL, BASIC, nous ferons une distinction moins subtile :

3 est une constante,
a, b, x, y sont des variables.

Les distinctions du mathématicien perdent leur intérêt lorsqu'il faut effectivement faire des calculs numériques. Par contre, elles présentent un intérêt pour les « manipulations algébriques » qui sont possibles avec des langages du type FORMAC qui sont peu répandus.

1.4.8. Représentation des nombres.

1.4.8.1. Rappel :

Le mathématicien classe les nombres en différentes catégories parmi lesquelles nous citerons :

— Les nombres entiers avec signe (cette classe se décompose en sous-classes).
— Les nombres rationnels (du type P/Q, P et Q étant des nombres entiers).
— Les nombres algébriques (solutions d'une équation polynominale : *exemple* $\sqrt{2}$.
— Les nombres transcendants (qui ne peuvent pas être solution d'une équation algébrique) : *exemple* π.
— Les nombres réels : ensemble des nombres algébriques et transcendants.

1.4.8.2. Représentation des nombres en ordinateur.

La représentation des nombres en machine se fait à partir d'un nombre de caractères limité qui ne permet pas de représenter des nombres quelconques.

En BASIC, nous nous intéresserons aux *nombres rationnels* ayant au plus n chiffres significatifs ; n dépend de l'ordinateur utilisé. Ces nombres seront représentés en virgule flottante.

Ce mode de représentation consiste à écrire tout nombre x sous forme de deux nombres, m et e, tels que :

$$X = m \times a^{e}$$

e ⟵ exposant ou caractéristique
a ⟵ base utilisée
m ⟵ mantisse

Exemple :

X = 312.124 est représenté : $X = 0.312124 \times 10^{3}$ (1).

La base a est :

10 pour l'être humain,
2 pour la plupart des machines de seconde génération.
16 pour les ordinateurs de troisième génération à vocation non exclusivement scientifique (cas des ordinateurs IBM370, Philips P1000 et UNIVAC série 90.

On peut aussi représenter tout nombre réel (au sens des mathématiciens) avec une erreur relative inférieure à une certaine valeur (de l'ordre de 10^{-6} à 10^{-15}) qui dépend du nombre de « bits » disponible pour représenter m.

Nous reviendrons sur ce mode de représentation au chapitre 2.

REMARQUE : Lorsqu'on fait des calculs à l'aide de la règle à calcul, on travaille avec un instrument qui donne toujours environ trois chiffres significatifs, l'exposant étant à déterminer (ou la position de la virgule) d'une autre manière.

Conséquences :

m ayant par exemple, au maximum sept chiffres significatifs, le calcul suivant :

$$X + (X \times 10^{-8})$$

donne le résultat X.

Il faut tenir compte de cette particularité dans l'analyse d'un problème : un procédé qui converge du point de vue mathématique *peut ne pas converger à cause des erreurs d'arrondi.*

1.4.8.3. Distinction entre virgule flottante normalisée et non normalisée.

1) Il y a lieu d'effectuer la distinction entre virgule flottante normalisée et virgule flottante non normalisée.

(1) Dans les notations anglo-saxonnes, la virgule qui sépare la partie entière de la partie fractionnaire est remplacée par un point.

Un nombre est exprimé en virgule flottante normalisée si le premier chiffre $\neq$ de 0 est celui qui suit immédiatement le point décimal.

Exemple : 0.125×10^2 est normalisée
1.25×10^1 n'est pas normalisée.

Quand on entre des données, on peut les écrire sous l'une des formes, normalisées ou non. Par contre, lorsque l'ordinateur imprime des variables en virgule flottante celles-ci sont toujours imprimées sous forme normalisée (sauf spécification contraire).

2) Dans la représentation interne en langage machine, la forme normalisée est celle qui permet à encombrement égal d'avoir le maximum de chiffres significatifs.

1.4.9. Objet de la compilation.

Les langages de programmation doivent satisfaire à certaines règles en particulier à des *règles de syntaxe* qui ressemblent aux *règles de grammaire* des langues naturelles (français, allemand, etc.).

Cependant, les langages de programmation sont beaucoup moins souples et le programmeur doit observer strictement ces règles.

Nous avons vu que le compilateur est un programme particulier, fourni par le constructeur d'ordinateurs, programme qui, à partir d'un programme écrit en « langage évolué » (dit « programme source »), produit un programme en langage machine (dit « programme objet »).

Lorsqu'un programme source ne satisfait pas aux règles de syntaxe, le compilateur ne peut pas effectuer la traduction en « langage objet ». Il donne alors des indications sur le type d'erreur et leur localisation. Le programmeur peut ainsi trouver plus rapidement les fautes commises et les corriger.

Le compilateur vérifie, en général, que le programme source satisfait aux conditions suivantes :

— les instructions sont correctes du point de vue morphologique et syntaxique (1) ;

— le programme source ne contient pas plusieurs instructions ayant la même étiquette ;

— aucune instruction ne fait référence à une étiquette absente.

Le travail du compilateur peut donc être représenté schématiquement par l'organigramme suivant (fig. 1.7) :

(1) Les fautes morphologiques correspondent sensiblement aux fautes d'orthographe dans les langues naturelles.

Les fautes syntaxiques correspondent aux fautes de grammaire.

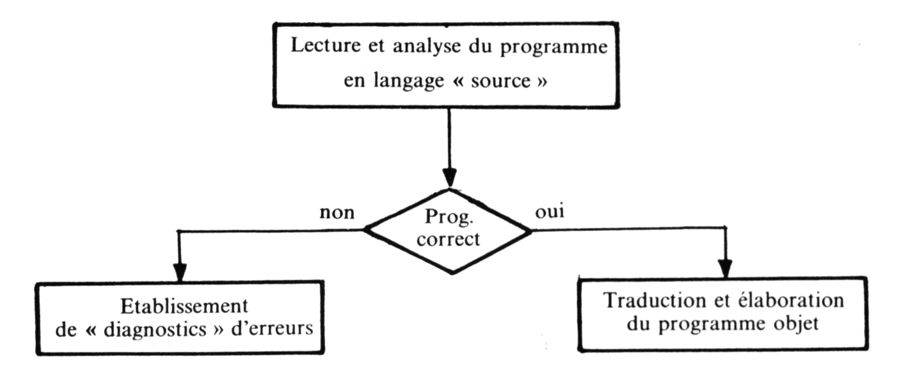


FIG. 1.7.

Il existe différentes méthodes pour réaliser les compilateurs : le choix de la méthode dépend du langage à traduire, du type d'ordinateur utilisé, ainsi que du mode d'exploitation envisagée.

1.4.9.1. Notion de priorité des opérateurs.

Considérons l'expression suivante :

$$z + a \sin x - \frac{b}{c}$$

que nous écrirons

$$Z + A * SIN(X) - B/C.$$

Pour calculer la valeur numérique de cette expression, il faut au préalable calculer A × SIN (X) et B/C et ensuite effectuer l'addition et la soustraction.

Pour permettre au compilateur de déterminer l'ordre sous lequel il faut effectuer les calculs, il est attribué une PRIORITÉ aux opérateurs. Le tableau de priorité suivant, donné à titre d'exemple, sera utilisé par les compilateurs BASIC :

Désignation des opérateurs	*Notation*	*Priorité par ordre décroissant*
Élévation à la puissance	↑	1
Multiplication et division	* /	2
Addition et soustraction	+ −	3

FIG. 1.8.

Ce tableau qui correspond sensiblement à celui du BASIC peut varier d'un langage à l'autre (1).

L'algorithme donnant l'ordre dans lequel les calculs doivent être effectués a l'allure suivante :

— Calculer d'abord chaque « sous expression » entre parenthèses. Pour cela on commence par la sous-expression située entre les parenthèses les plus intérieures (on dit aussi les plus *profondes*).

— A profondeur égale, évaluer la sous-expression située le plus à gauche dans l'expression.

— Pour évaluer une expression ou sous-expression ne contenant aucune parenthèse, l'ordinateur :

● exécute les opérations les plus prioritaires (en utilisant le tableau précédent) et à priorité égale, commence par les opérations situées le plus à gauche..

Exemple : A + B * SIN (X) – (3 * X ↑ 2)/5.

1[re] *étape.* Calcul de (3 * X ↑ 2) : pour cela calculer des résultats intermédiaires R1 et R2

R1 = X ↑ 2
R2 = 3 * R1

2[e] *étape.* Calcul de B * SIN (X) et des résultats intermédiaires R3 et R4.

R3 = SIN (X)
R4 = B * R3

3[e] *étape.* Calcul de (3 * X ↑ 2)/5 à partir de R2

R5 = R2/5.

4[e] *étape.* Addition et soustraction

A + R4 – R5.

En fait il y a quelques variantes à ce procédé en particulier dans les compilateurs très évolués qui doivent effectuer une sorte d'optimisation.

1.5. Annexe : description de la syntaxe d'un langage.

Pour pouvoir définir avec précision et sans ambiguïté possible la syntaxe d'un langage, il faut pouvoir disposer d'une méthode rigoureuse de description.

(1) Ce tableau est immuable à l'intérieur d'un langage simple, c'est-à-dire, que l'utilisateur ne peut pas modifier ce tableau.

Ce n'est pas le cas du langage ALGOL-68 qui offre à l'utilisateur la possibilité de définir ses propres opérations et leur degré de priorité.

Pour les langages BASIC, FORTRAN, ALGOL, on utilise la *Forme Normale de Backus* ou la carte *syntaxique* qui est équivalente à la forme normale de Backus.

1.5.1. Notion de carte syntaxique.

Une carte syntaxique est formée à partir de différents symboles :
— des cercles qui contiennent les symboles de base du langage,
— des ovales qui contiennent des notions qui sont définies à cet endroit,
— des rectangles qui contiennent des notions qui sont définies ailleurs.

La carte de la figure 1.9 indique que la notion V peut être soit le symbole de base U, soit la notion W suivie du symbole de base U mais V ne peut pas être la notion W seule.

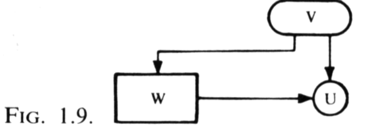


FIG. 1.9.

Cette notion W est supposée être définie ailleurs (puisqu'elle est écrite dans un rectangle).

Exemple : Nous pouvons définir un nombre entier de la façon suivante : un nombre peut être :
— soit un entier sans signe,
— soit un signe suivi d'un entier sans signe.
Ceci donne le schéma suivant :

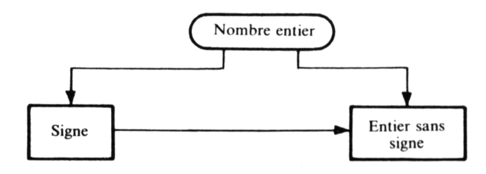


FIG. 1.10.

Nous savons en outre qu'un signe est constitué de l'un des symboles de base suivants : le signe $-$ ou le signe $+$.

Un entier sans signe est constitué d'une suite de chiffres, chaque chiffre étant un symbole de base.

Un nombre entier peut donc être défini par la carte suivante :

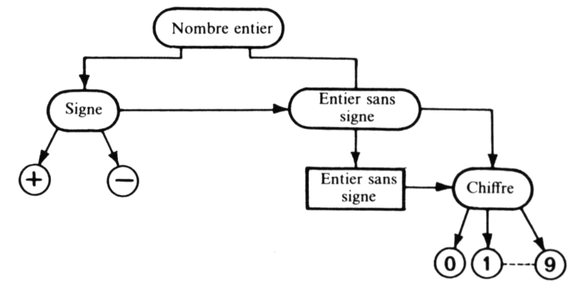


FIG. 1.11.

1.5.2. Forme normale de Backus.

Cette forme permet de représenter la syntaxe sous forme linéaire.

Prenons l'exemple précédent :

⟨ nombre entier ⟩ : : = ⟨ signe ⟩ ⟨ entier sans signe ⟩ | ⟨ entier sans signe ⟩
⟨ entier sans signe ⟩ : : = ⟨ entier sans signe ⟩ ⟨ chiffre ⟩ | ⟨ chiffre ⟩
⟨ chiffre ⟩ : : = 0|1|2|3|4|5|6|7|8|9
⟨ signe ⟩ : : – | +

Dans cet exemple les notions sont encadrées par les symboles ⟨ et ⟩, la barre verticale signifie l'un ou l'autre

Lorsque deux notions se suivent sans être séparées par une barre, cela signifie que la première doit être suivie de la seconde, par exemple, dans la première ligne, le signe doit être suivi d'un entier sans signe.

1.6. Exercices.

1.6.1. Calcul du nombre e.

Rappelons que la valeur de e peut s'obtenir par le calcul de la série :

$$e = 1 + \frac{1}{1\,!} + \frac{1}{2\,!} + \frac{1}{3\,!} + \frac{1}{4\,!} + \ldots$$

compte tenu du fait que cette série converge vite et que la précision de l'ordinateur utilisé est limitée, on calculera :

$$s_n = \sum_{i=1}^{n} \frac{1}{i\,!}$$

Premier cas :

On arrêtera les calculs lorsque les erreurs d'arrondi feront que :

$$s_n = s_{n-1}.$$

Il y aura lieu d'imprimer la valeur de e ainsi obtenue et le nombre de termes de la série utilisée.

Construire l'organigramme.

Méthode : On utilisera trois variables E, F, X de la façon suivante :

F contient $\frac{1}{i\,!}$ et contiendra au pas suivant $\frac{1}{(i+1)\,!}$

E contient S_i et on obtiendra S_{i+1} dans la variable X.

Si X ≠ E, alors on transfère la valeur X dans E et on continue le calcul.

Si X = E, alors on arrête le calcul

Deuxième cas :

Pour améliorer la précision, la somme des termes de la série sera faite « à rebours ». Déterminer dans ce cas l'organigramme.

1.6.2. Recherche de nombres premiers.

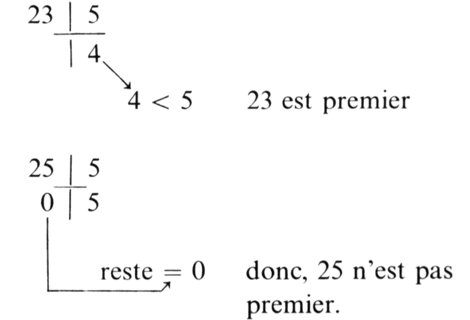
Les nombres 1, 2, 3 étant connus comme étant premiers, les autres nombres premiers sont tous de la forme $6n \pm 1$, n étant entier positif. Pour savoir si un nombre est premier, on le divise par les nombres premiers précédemment trouvés, jusqu'à ce que le quotient devienne inférieur au diviseur.

Si aucune division ne donne un reste nul, le nombre est premier.

Par ailleurs, les nombres de la forme $6n + 1$ ne sont ni pairs ni multiples de 3, il n'y a donc pas lieu de faire les divisions ni par 2, ni par 3 (ceci entraîne une petite difficulté pour 5).

Exemple :

$n = 1$	donne	5 et 7
$n = 2$	»	11 et 13
$n = 3$	»	17 et 19
$n = 4$	»	23 et 25



1.6.3. Solutions.

Nous utiliserons maintenant le symbolisme :

x = expression, le caractère = étant un symbole d'affectation.

Premier cas

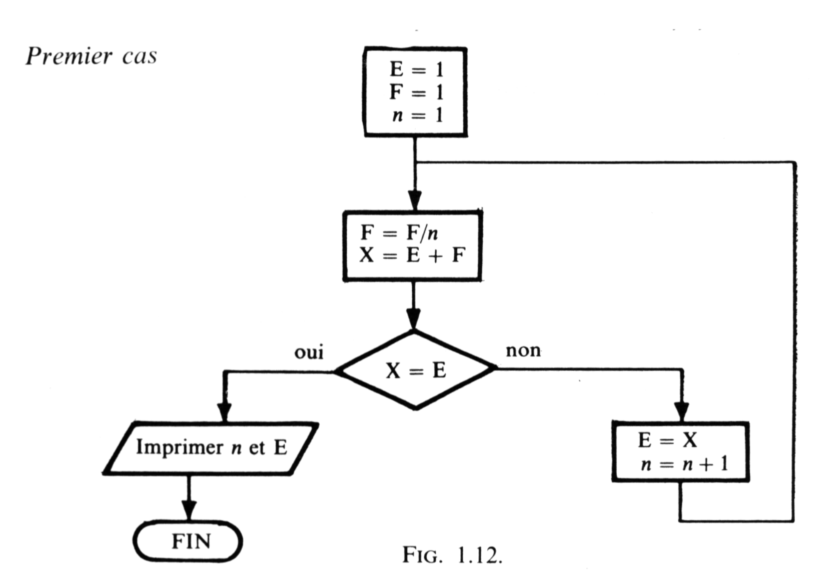


FIG. 1.12.

Deuxième cas

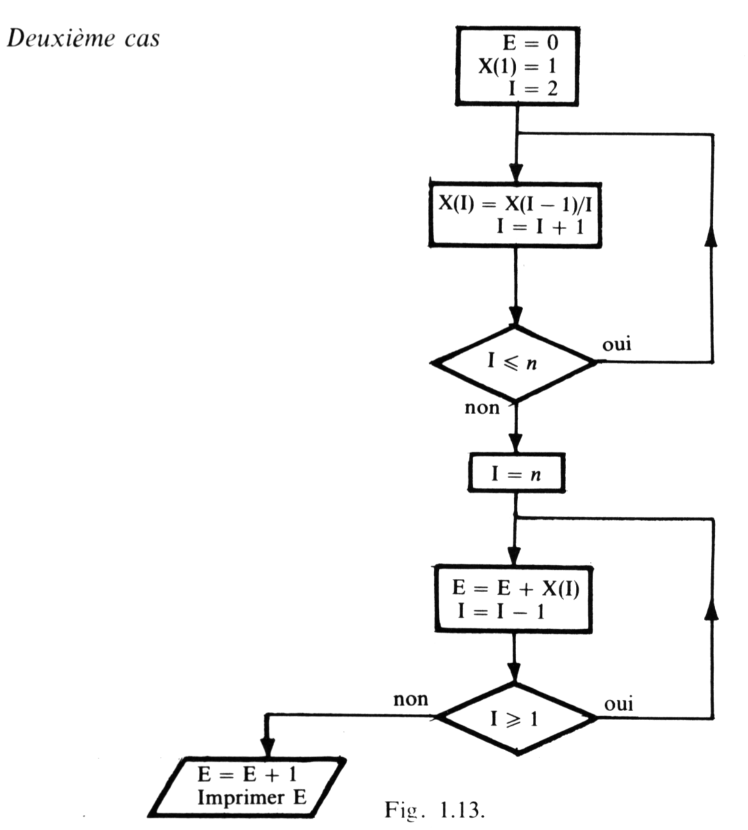


Fig. 1.13.

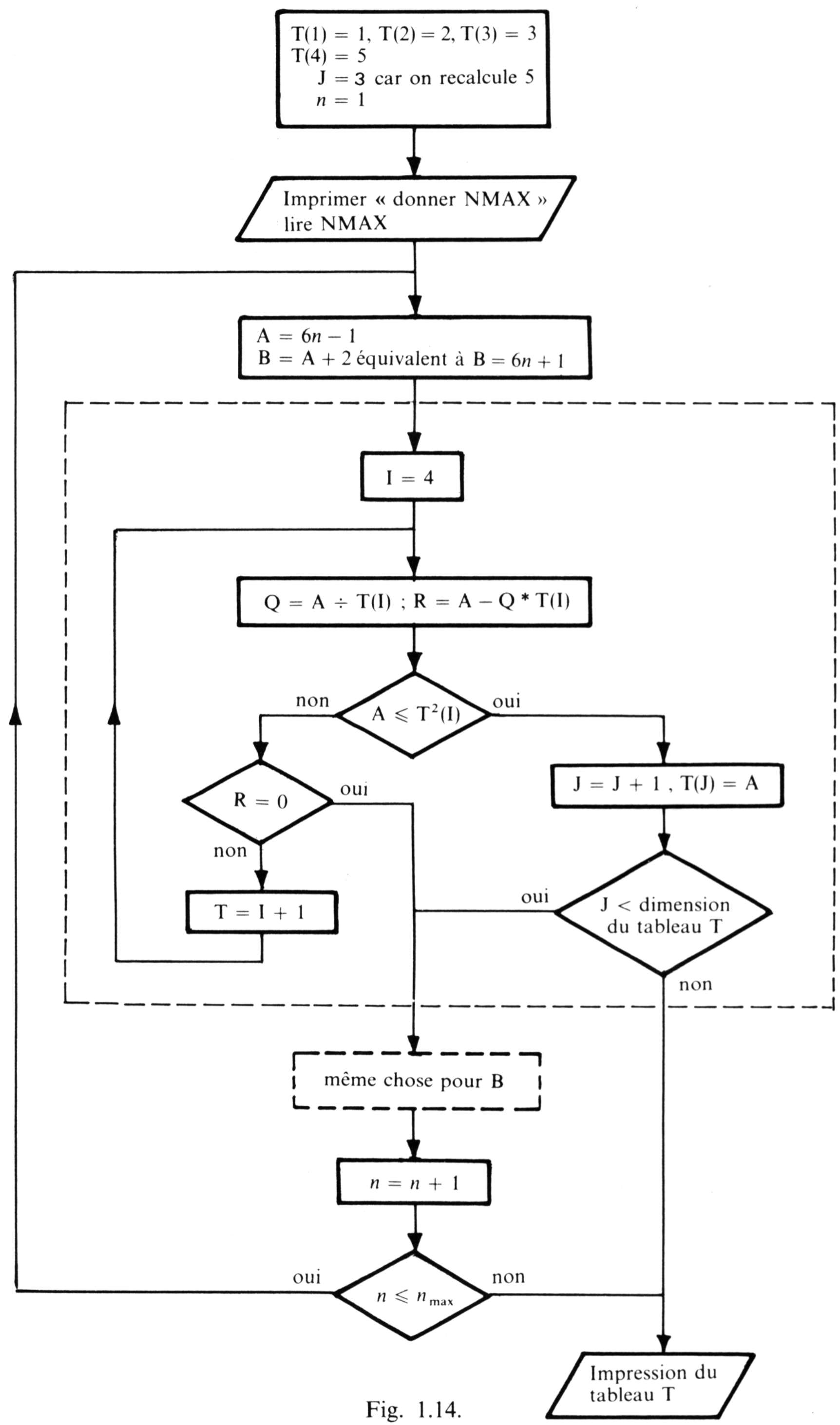


Fig. 1.14.
Cet organigramme initial pourra être remplacé par celui de la figure 1.14 *bis*.

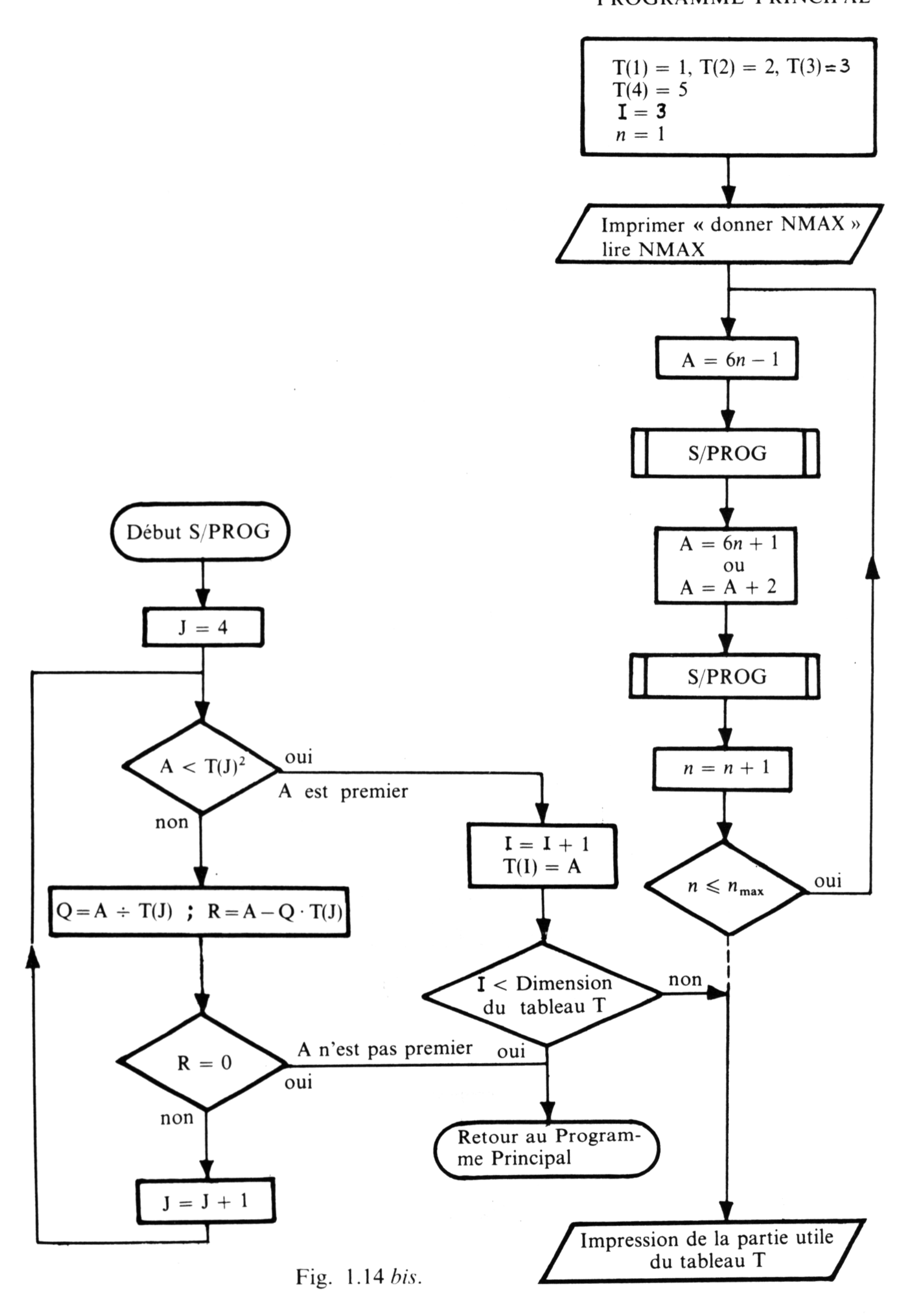

PROGRAMME PRINCIPAL
T(1) = 1, T(2) = 2, T(3) = 3
T(4) = 5
I = 3
n = 1
Imprimer « donner NMAX »
lire NMAX
A = 6n − 1
S/PROG
A = 6n + 1
ou
A = A + 2
S/PROG
n = n + 1
n ⩽ n max
oui
non
Impression de la partie utile
du tableau T
Début S/PROG
J = 4
A < T(J)²
oui
A est premier
non
I = I + 1
T(I) = A
Q = A ÷ T(J) ; R = A − Q · T(J)
I < Dimension
du tableau T
non
R = 0
A n'est pas premier
oui
oui
non
Retour au Programme Principal
J = J + 1

Fig. 1.14 *bis*.

CHAPITRE 2

LES ÉLÉMENTS DU LANGAGE BASIC

2.0. Introduction : orientation du langage basic.

Le BASIC a été conçu par le Dartmouth College afin de permettre un accès rapide et facile à l'ordinateur pour de nombreux étudiants. Le sigle de BASIC vient de « Beginner's All purpose Symbolic Instruction Code ».

Les impératifs qui ont guidé la conception du langage sont :

— facilité d'apprentissage,

— facilité et rapidité de compilation du langage,

— facilité d'utilisation à partir de terminaux de « Time-Sharing »,

— ouverture vers des applications scientifiques et de gestion relativement simples,

— précision des diagnostics lors de la compilation et surtout lors de l'exécution du programme,

— possibilité d'être exploité à partir d'un ordinateur de taille très modeste.

Ces objectifs ont conduit les créateurs du BASIC à créer un langage simple dont les principales caractéristiques sont les suivantes :

— utilisation d'un vocabulaire restreint

— adjonction systématique d'une étiquette à chaque instruction sous forme de numéro de ligne (une seule instruction par ligne)

— pour le calcul : utilisation d'un seul type de variables (variables réelles représentées en virgule flottante)

— suppression de sous-programmes externes, ce qui entraîne certaines complications (certains constructeurs n'ont pas accepté cette restriction issue de la première version du BASIC).

Depuis sa conception, pour pallier certaines insuffisances, ce langage a subi diverses extensions qui varient d'un compilateur à l'autre et dont les principales sont signalées au chapitre 11.

2.1. Alphabet disponible en basic.

Comme tous les langages, le BASIC est construit à partir d'un ensemble de symboles de base (dit ALPHABET). Cet ensemble est ici très simple mais varie un peu d'un constructeur à l'autre. Le tableau suivant donne les alphabets les plus couramment utilisés.

Afin de distinguer le chiffre zéro et la lettre O, nous barrons l'un des symboles. En outre, le caractère blanc sera représenté soit par un blanc, soit par le symbole ␣.

Désignation des caractères	ASC II	EBCDIC
— Les chiffres de 0 à 9	oui	oui
— Les lettres majuscules A, B, ..., Z	oui	oui
— Les symboles opératoires :		
• moins	−	−
• plus	+	+
• multiplication	*	*
• division	/	/
• élévation à la puissance	↑	↑
— Les parenthèses :		
• gauche	(	(
• droite	)	)
— Les caractères de comparaison :		
• égal	=	=
• inférieur à	<	<
• supérieur à	>	>
• inférieur ou égal	< =	⩽
• supérieur ou égal	> =	⩾
• différent de	< >	≠
— Les caractères de ponctuation :		
• point	.	.
• virgule	,	,
• point virgule	;	;
• point d'exclamation	!	!
• point d'interrogation	?	?
• deux points	:	:
— Les caractères spéciaux suivants :		
• simple guillemet	'	'
• doubles guillemets	"	"
• dollar	$	$
• pour cent	%	
• et commercial	&	ε
• a « escargot »		@
• « dièse »		#
• blanc		

FIG. 2.1.

Cette distinction s'explique par le fait que certains systèmes utilisent le code ASCII (1) à 7 bits alors que le Call 360 utilise le code EBCDIC (1) à 8 bits qui donne par conséquent quelques possibilités supplémentaires.

Sur les systèmes utilisant le code ASCII, les caractères ⩾, ⩽, et ≠ sont remplacés respectivement par :

> =
< =
< >

Le système CALL 360 accepte aussi cette séquence (2).

Certains caractères tels que → et @ bien que disponibles en ASCII sont réservés à un usage particulier et leur utilisation en BASIC est soumise à certaines restrictions.

Tout le langage est construit à partir de cet alphabet et comporte essentiellement les notions suivantes :

— des *mots-clés* destinés à construire les INSTRUCTIONS,
— des variables,
— des constantes,
— des fonctions de base.

2.2. Structure générale d'un programme basic. Les étiquettes.

En BASIC, nous écrirons une instruction par ligne et chaque ligne comportera au début un numéro de ligne, ce numéro tiendra lieu d'étiquette.

Exemple :

```
10 INPUT A, B
20 LET  X = A + B
     .
     .
     .
60 END
```

Les numéros de ligne iront en croissant mais ne doivent pas forcément respecter une progression régulière.

La dernière instruction d'un programme doit être END.

La numérotation systématique des lignes permet une modification simple et aisée du programme à partir d'un terminal de Time Sharing.

(1) ASCII : American Standard Code for Information Interchange
EBCDIC : Extended Binary Coded Decimal Interchange Code
Le code ASCII est très utilisé pour les transmissions à basse vitesse.

(2) Certains systèmes acceptent aussi l'écriture inversée : = >, = < et ≠ .

2.3. Les constantes et les variables.

Le BASIC accepte deux types de variables et de constantes :

— les variables et constantes « numériques » qui sont représentées en virgule flottante.

— les variables et constantes « caractères ». Cette dénomination correspond au terme américain « STRING » dont la traduction française est « chaîne » ou « alphamérique » (ce dernier mot vient d'IBM-FRANCE) ou « caractères ».

Ces variables peuvent être des variables simples ou des variables indicées.

2.3.1. Les constantes et variables réelles.

Une variable RÉELLE est représentée soit par une lettre, soit par une lettre suivie d'un chiffre (chiffre de 0 à 9).

Exemple :

A, B
A3, X9] sont des variables réelles

A12
XM] ne sont pas admises par BASIC

Le BASIC dispose donc pour effectuer des calculs de

$26 \times 11 = 286$ noms de variables réelles (1).

Une variable réelle peut être *indicée* et avoir un ou deux indices. Toutefois, le BASIC de certains systèmes accepte des variables à trois indices (cf. 11.1).

Un identificateur de *variable indicée* ne doit comporter qu'une *seule lettre*. Ceci réduit au sein d'un même programme le nombre maximum de tableaux possibles à

26 pour le code ASCII
29 pour le code EBCDIC (CALL 360).

Une variable peut être indicée de deux façons :

— si le ou les indices varient de 0 à 10, il n'est pas nécessaire de préciser la plage de variation du ou des indices et un dimensionnement « impli-

(1) Ceci suffit pour de nombreuses applications. Cependant les langages habituels (FORTRAN, COBOL, etc.) acceptent des noms de variables plus longs (six caractères alphanumériques pour le FORTRAN), ce qui est beaucoup plus commode.

cite » à 10 est réalisé dès que l'on rencontre cette variable accompagnée d'un indice ;

— si le ou les indices ont une plage de variation différente, il faut au préalable déclarer les limites que peuvent atteindre ces indices au moyen de l'instruction DIM.

A condition de ne pas utiliser les instructions de traitement de tableaux, on peut se dispenser dans certains cas de déclarer des tableaux dont la plage de variation des indices est inférieure à 10. Cependant ceci est à déconseiller car l'absence de déclaration rend la mise au point des programmes moins commode ainsi que leur maintenance.

Exemple :

```
DIM A(10, 20), B(50)
```

Attention : Pour certains systèmes l'instruction OPTION BASE permet d'indiquer si les indices vont de 0 à *n* ou de 1 à *n* (cf. p. 87).

Les détails relatifs à l'instruction DIM sont indiqués en 7.1. Certaines extensions sont signalées en 11.1.2.

Une constante sera représentée sous l'une des trois formes suivantes :

— entière	exemple	– 325
— décimale	exemple	3.141592654
— décimale avec exposant	exemple	.31415926 E + 01 (entier)

Lorsque la constante est positive, le signe + est optionnel. Par contre pour représenter un nombre négatif, il faut obligatoirement le faire précéder du signe – .

2.3.2. Les constantes et variables caractères.

Ces éléments sont destinés à permettre de façon simple le traitement de caractères. Ces variables sont représentées par une lettre suivie du caractère dollar $.

Exemple : A$, X$ sont des variables caractères.

Nous disposons donc en BASIC d'au plus 26 variables caractères de noms différents, mais ces variables peuvent être indicées selon les mêmes règles que les variables réelles.

REMARQUE : Cette notion de variable et de constante « caractères » n'existait pas dans la version initiale de BASIC car elle n'est pas utile pour le calcul purement numérique.

Nous verrons cependant qu'elle permet d'étendre le domaine d'application du BASIC à des traitements de caractères et à des applications simples de gestion.

Une variable caractère peut contenir de zéro à *n* caractères, *n* dépendant de l'ordinateur utilisé. Le tableau suivant indique la longueur maximum pour les principaux systèmes BASIC.

Systèmes		*Longueur maximum*		
IBM	CALL 360	0	à	18
PHILIPS	TS 9200	0	à	72
CHB/General Electric	(Système MARK 2)	0	à	4095 (1)
CDC 6000		limite		élevée
Pdp 11				

FIG. 2.2.

REMARQUE : 72 caractères correspondent à la longueur maximum d'une ligne éditée sur un télétype normal. PHILIPS n'accepte pas de véritables variables caractères indicées, mais donne la possibilité à l'utilisateur de fixer la longueur qu'il désire. Cette longueur ne peut excéder 72 caractères.

Une variable caractère peut être indicée, mais certains systèmes n'acceptent qu'un seul indice.

Exemple : DIM A$ (12) , B$(15)
DIM X(20, 5) , X(15)

Une constante caractère s'écrit entre deux guillemets :

Exemple : " ABCD " désigne la constante caractère qui contient les quatre caractères ABCD.
" ABCD " désigne la constante caractère qui contient les cinq caractères ␣ABCD
␣ représentation du blanc.

REMARQUE : Lorsqu'une variable ne contient que *p* caractères alors qu'elle peut en contenir au total *n*, les $n - p$ « casiers » libres sont en général remplis de « blanc ».

— En général, les variables caractères sont initialisées avec des « blancs ».

L'utilisation des variables caractères est exposée au chapitre 8.

2.3.3. Utilisation des variables indicées.

Pour faire référence à une variable indicée, par exemple à A(5), dans une expression arithmétique, ou une instruction de sortie nous écrirons . . . A(5) . . .

(1) En entrée la limite est de 158 caractères.

Mais nous pouvons également utiliser comme indice une variable

I = 5

..... A (I) ou A (I + 3 * X)

Normalement la valeur de la variable ou même de l'expression qui tient lieu d'indice doit être *entière*. Si tel n'est pas le cas, la majorité des interpréteurs BASIC effectuent une conversion en entier par TRONCATURE. Cependant le projet de nouvelle norme (septembre 1979) demande à ce que la conversion soit effectuée par ARRONDI.

Exemple :

Solution en vigueur	Solution demandée
110 I = 12.8	110 I = 12.8
130 X = X + A(I)	130 X = X + A(J)
en 130 on prend A(12)	en 130 on prend A(13)

2.4. Les fonctions standards.

L'utilisateur peut faire référence à certaines fonctions disponibles en BASIC sans avoir à les définir au préalable. Ces fonctions, appelées *fonctions standards* peuvent se classer en trois catégories :

— fonctions mathématiques usuelles,

— fonctions liées au système; abordées en 2.4.2 elles sont étudiées au chapitre 9,

— fonctions pour « chaînes de caractères » étudiées au chapitre 8.

2.4.1. Les fonctions mathématiques usuelles.

Les tableaux des figures 2.3 et 2.3 *bis* donnent la liste des fonctions disponibles sur la plupart des systèmes et des fonctions disponibles de façon moins fréquente.

D'autres fonctions liées à certaines extensions (variables entières et surtout variables réelles) sont signalées au chapitre 11.

2.4.1.1. Fonction ATN.

Y = ATN(X) donne Y en radians et situé dans l'intervalle

$$\left(-\frac{\pi}{2}, +\frac{\pi}{2}\right).$$

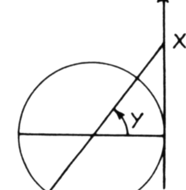


FIG. 2.4.

Significations	*Noms*	*Commentaires*
Racine carrée de X	SQR(X)	
Sinus de X	SIN(X)	L'angle X doit être donné en radians
Cosinus de X	COS(X)	
Tangente de X	TAN(X)	
Arctangente de X	ATN(X)	Donne un résultat en radians situé dans l'intervalle $-\frac{\pi}{2} + \frac{\pi}{2}$. Cette fonction est appelée ATAN sur certains systèmes (Pdp 11 par ex.).
Exponentielle de X	EXP(X)	
Valeur absolue de X	ABS(X)	
Entier de X	INT(X)	Plus grand entier inférieur ou égal à X.
Signe de X	SGN(X)	Donne pour résultat 1 si $X > 0$, 0 si $X = 0$, -1 si $X < 0$.
Logarithme naturel de X	LOG(X)	
Logarithme base 10 de X	LGT(X)	
Logarithme base 2 de X	LTW(X)	Disponible sur le système CALL 360.
Nombre aléatoire entre 0 et 1	RND(X)	X est sans signification mais nécessaire pour des raisons de syntaxe.
Minimum de	MIN(A, B, C...)	Seules ces deux fonctions lorsqu'elles sont disponibles (systèmes PHILIPS par exemple) acceptent plusieurs arguments.
Maximum de	MAX(A, B, C...)	
Tabulation	TAB(X)	Fonction particulière liée au système (voir paragraphe 5.4.2).

FIG. 2.3.

REMARQUE : Cette fonction est la même que la fonction ATAN du FORTRAN qui dispose en outre de la fonction ATAN2 qui est très commode (1).

Lorsqu'il y a lieu de lever l'indétermination, il faut faire appel à d'autres informations ; voir à titre d'exemple l'exercice relatif au volume d'une cuve cylindrique.

2.4.1.2. Fonction SQR :

Y = SQR(X) donne pour Y une valeur positive ou nulle

SQR (4) donne 2
SQR (– 4) donne un diagnostic d'erreur.

(1) Cf. *Le langage FORTRAN IV*, bibliothèque technique PHILIPS, même auteur.

Significations	*Noms*	*Commentaires*
Les fonctions suivantes ne sont disponibles que sur certains systèmes (notamment CALL 360 et VSPC)		
Cotangente de X	COT(X)	
Sécante de X	SEC(X)	
Cosécante de X	CSC(X)	
Arcsinus de X	ASN(X)	Le système XDS 940 utilise d'autres noms qui sont ASIN et ACOS
Arcosinus de X	ACS(X)	
Sinus hyperbolique de X	HSN(X)	
Cosinus hyperbolique de X	HCS(X)	
Tangente hyperbolique de X	HTN(X)	
Valeur en degré de X en radian	DEG(X)	
Valeur en radian de X en degré	RAD(X)	
Arrondi de X	ROUN(X)	Donne pour résultat l'entier le plus proche de X. Disponible sur XDS 940.
Partie entière de X	FIX(X)	Disponible notamment sur XDS 940.
Partie fractionnaire de X	FP(X)	
Déterminant	DET	La syntaxe de cette fonction varie d'un système à l'autre disponible sur certains systèmes (MARK II, pdp 11, Data General, etc.).

Fig. 2.3 *bis*.

2.4.1.3. Fonction ABS :

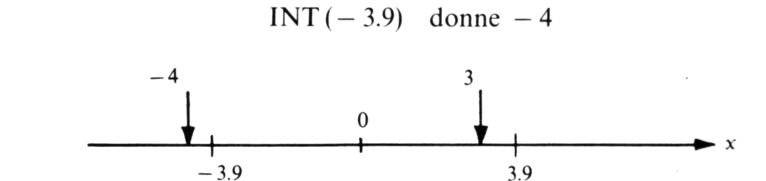
Cette fonction donne la valeur absolue d'un nombre.

Exemple : ABS (3.5) donne 3.5
ABS (– 3.5) donne 3.5

2.4.1.4. Fonction INT :

Le plus grand entier inférieur ou égal à X ne se calcule pas de la même manière selon que X est positif ou négatif.

Exemple : INT (3.9) donne 3
INT (– 3.9) donne – 4



2.4.1.5. Fonction ROUN :

Cette fonction assez rarement disponible effectue l'arrondi :

ROUN (3.9) donne 4
ROUN (– 3.9) donne – 4
ROUN (3.3) donne 3
ROUN (– 3.3) donne – 3

2.4.1.6. Fonctions FIX et FP :

Ces deux fonctions permettent de décomposer un nombre en partie entière et partie fractionnaire.

FIX (3.9) donne 3
FP (3.9) donne 0.9

sous cette forme $FP(x) = X - FIX(x)$
$= X - INT(x)$

mais si l'argument est négatif, le résultat est le suivant :

FIX (– 3,9) donne – 4
FP (– 3,9) donne + 0,1

2.4.1.7. Fonctions MIN et MAX :

Ces fonctions acceptent au moins deux arguments :

MIN (3,4) donne pour résultat 3
MIN (– 10,3) donne pour résultat – 10
MAX (3,4) donne pour résultat 4
MAX (– 10,4) donne pour résultat 4

2.4.1.8. Fonctions RND :

Ici l'argument X est nécessaire uniquement pour des raisons syntaxiques mais n'a aucune influence sur le résultat : cette fonction donne pour résultat un nombre pseudo-aléatoire uniformément réparti dans l'intervalle [0, 1].

REMARQUE

1) Il s'agit uniquement de nombres « PSEUDO-ALÉATOIRES ». En effet, ils sont créés par un sous-programme qui est forcément « déterministe ». Lorsqu'on exécute plusieurs fois un même programme faisant appel à cette fonction, les deux suites de nombres produits sont les mêmes.

2) Certains systèmes (pdp 11) disposent d'une instruction supplémentaire RANDOMIZE qui, placée en tête de programme fait partir d'une valeur de départ différente et permet ainsi d'obtenir à chaque exécution une suite de nombres aléatoires différents.

3) La fonction RND a une interprétation différente sur certains systèmes (notamment XDS 940) :

RND (0) correspond à la définition,
RND (nombre négatif) donne un nombre déterminé à partir de l'horloge interne de l'ordinateur,
RND (nombre positif) donne un nombre non aléatoire qui dépend de l'argument.

2.4.2. Généralités sur les fonctions liées au système.

Il peut être intéressant au cours de l'exécution d'un programme de disposer de certaines informations telles que, la date, le temps unité centrale consommée depuis le début de l'exécution du programme, etc. La disponibilité de telles informations est fortement liée au système d'exploitation de l'ordinateur. En général on disposera d'une fonction DATE pour les systèmes acceptant les chaînes de caractères et de la fonction TIME.

D'autres fonctions peuvent cependant être envisagées.

Le chapitre 9 expose ces différentes fonctions.

2.5. Exercices.

Faire l'organigramme du calcul de π par le calcul de la somme des séries suivantes :

$$\frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \dots$$

$$\frac{\pi^2}{8} = 1 + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \dots$$

$$\frac{\pi^2}{12} = 1 - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \dots$$

$$\frac{\pi^4}{90} = 1 + \frac{1}{2^4} + \frac{1}{3^4} + \frac{1}{4^4} + \dots$$

On arrête le calcul dès que l'une des conditions suivantes est satisfaite :

— on a utilisé 40 000 termes,

— la précision ne s'améliore pas.

Pour cela, on fera une comparaison entre S_n et S_{n+1} et on arrête dès que $S_n = S_{n+1}$

$$S_n = \sum_{i=n}^{n} u_n$$

— Faire l'organigramme d'un programme de calcul d'amortissement.

Solutions :

— Calcul de π à partir de la série donnant $\frac{\pi^2}{6}$, $\frac{\pi^2}{8}$, $\frac{\pi^2}{12}$ et $\frac{\pi^4}{90}$.

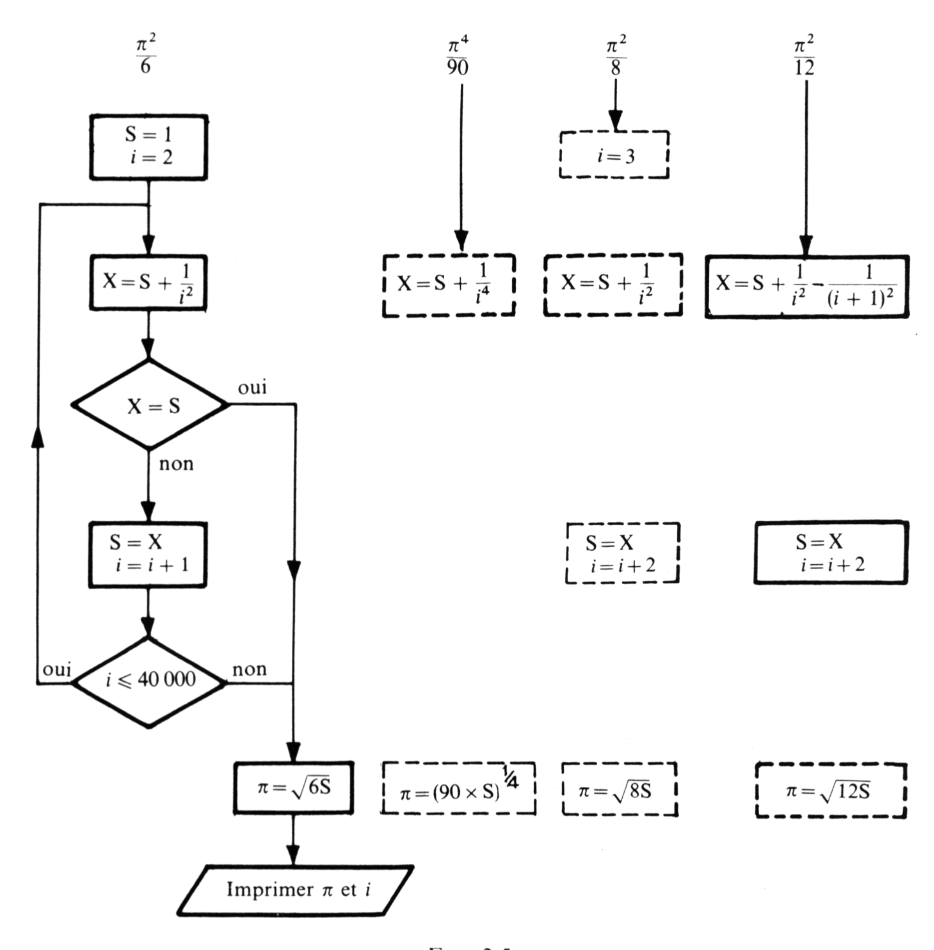


FIG. 2.5.

— Calcul de π à partir de la série $\frac{\pi^4}{90}$. L'organigramme a la même allure à l'exception des deux rectangles en traits interrompus.

— Calcul de π à partir de la série $\frac{\pi^2}{8}$: la principale différence consiste dans le fait que la variable i est incrémentée par pas de 2

— Calcul de π à partir de la série $\frac{\pi^2}{12}$:

puisque la série est alternée, on peut :

— ou bien décomposer le calcul de façon à tenir compte des termes positifs et des termes négatifs,

— ou bien mémoriser le signe du terme d'indice i pour déterminer le signe du terme $i + 1$,

— ou bien rassembler deux termes consécutifs en un seul de façon à constituer une série non alternée :

$$\frac{\pi^2}{12} = \left(\frac{1}{1^2} - \frac{1}{2^2}\right) + \left(\frac{1}{3^2} - \frac{1}{4^2}\right) + \left(\frac{1}{5^2} - \frac{1}{6^2}\right) + \ldots$$

Cette dernière méthode est la plus simple et conduit à une boucle de calcul dont le pas est 2 :

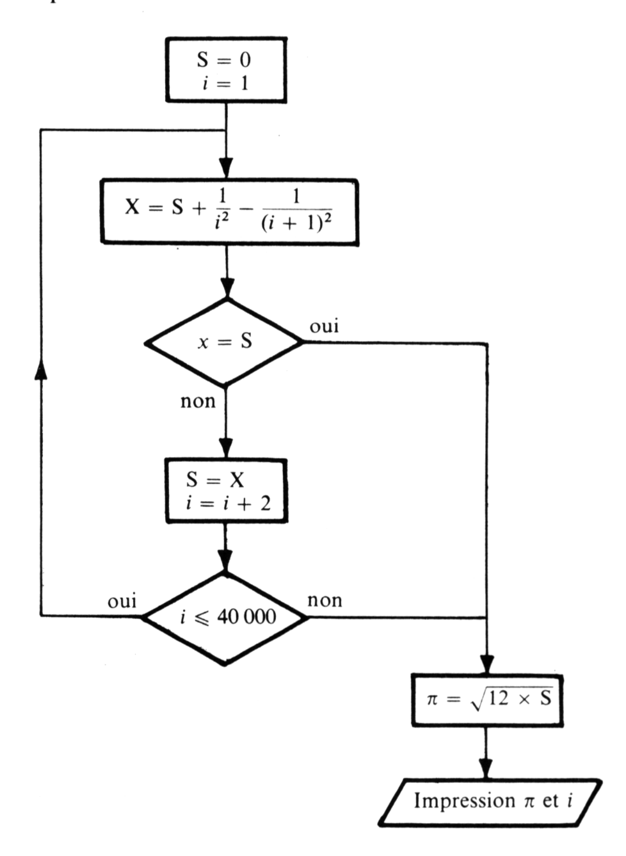


FIG. 2.6.

CHAPITRE 3

LES INSTRUCTIONS D'AFFECTATION, DE TEST ET DE CONTRÔLE

3.0. Introduction.

Dans ce chapitre, nous étudierons les instructions de traitement portant sur des variables réelles simples ou des éléments de tableaux réels. Par la suite nous constaterons que le BASIC offre pour le traitement des tableaux (vecteurs et matrices) des instructions plus synthétiques (cf. chap. 7).

3.1. L'instruction d'affectation arithmétique.

Cette instruction permet d'affecter la valeur numérique d'une expression arithmétique à une ou plusieurs variables (affectation simple ou affectation multiple). Sa forme est la suivante :

LET (1) $Var_1 = Var_2 = ... = Var_n =$ expression arithmétique

où Var_1 Var_2, ...Var_n sont des variables simples ou des éléments de tableau

Exemples :

LET (1) X = Y = 0	0 est affecté à X et Y.
LET (1) X = A = SIN (3 * Z)	la valeur numérique de SIN (3 * Z) est affectée à X et A.
LET (1) X = X + 3	la valeur de X + 3 est affectée X.

L'expression arithmétique est évaluée selon la méthode expliquée au paragraphe précédent.

(1) Le préfixe LET indispensable jadis est maintenant optionnel sur la plupart des compilateurs commercialisés.

REMARQUE : L'opérateur puissance n'étant pas associatif, l'expression A↑ B↑ C doit obligatoirement être évaluée selon un ordre déterminé (de gauche à droite) donc sous la forme

(A↑ B)↑ C.

Ce n'est pas le cas du langage PL/1 qui, à priorité égale, fait évaluer les expressions
— dans l'ordre de gauche à droite pour les opérateurs autres que puissance
— dans l'ordre de droite à gauche pour l'opérateur puissance.

3.2. Initialisation des variables.

Lors de l'exécution d'un programme BASIC, la valeur numérique des variables peut être initialisée de différentes manières :
— ou bien par lecture de données,
— ou bien par calcul à partir d'autres variables et constantes.

On peut aussi faire référence dans une expression arithmétique à une variable dont la valeur numérique n'a pas encore été déterminée au moyen de l'une des méthodes précédentes. Dans ce cas, certains systèmes attribuent la valeur 0 à cette variable et continuent l'exécution du programme (ex. GE 265).

D'autres systèmes tels que PHILIPS, HEWLETT PACKARD donnent le diagnostic « VALUE NOT ASSIGN » et stoppent l'exécution du programme. Certains compilateurs permettent encore plus de souplesse (cf. chap. XI).

Exemple : (numéros de lignes)

```
10   X = 3
20   Z = Z + Y
```

Ceci donne à l'exécution,

pour la plupart des systèmes	pour les systèmes PHILIPS
X = 3	X = 3
Z = 3 + 0	20 VALUE NOT ASSIGN

La première méthode est très commode car, très souvent, il faut initialiser les variables à zéro, mais peut se révéler dangereuse dans le cas où l'on a oublié d'initialiser une variable à une valeur autre que zéro.

La deuxième méthode, plus lourde dans certains cas, est plus sûre pour le débutant.

Un excellent compromis est disponible sur certains systèmes (cf. 11.1.3).

Remarque

En FORTRAN, aucune règle ne précise comment sont initialisées les variables si l'utilisateur a oublié de le faire.

L'ordinateur prend en général le contenu du « casier correspondant » lors de la fin du travail précédent.

3.3. Évaluation des expressions arithmétiques.

Les expressions arithmétiques du BASIC sont particulièrement simples et s'évaluent comme il est indiqué dans le chapitre 1.

Lorsque l'on n'est pas sûr de l'ordre dans lequel sera évaluée l'expression, on peut toujours préciser cet ordre en ajoutant des parenthèses.

Des parenthèses inutiles ne constituent pas une faute syntaxique.

3.4. Exercices.

Écrire en BASIC les expressions suivantes en s'aidant du tableau des fonctions standard :

$$y = a \sin (x + \pi/3)$$

$$y = e^{-\frac{x2}{2}}$$

$$z = \sqrt{x^2 + y^2 \sin (x)}$$

$$z = \text{entier (Log (X + 5))}$$

Réponses :

```
Y=A*SIN(X+3.1415926/3)
```

```
Y=EXP(-X↑2/2)
```
ou
```
Y=EXP(-X*X/2)
```

```
Z=SQR(X↑2+Y↑2*SIN(X))
```
ou
```
Z=SQR(X*X+Y*Y*SIN(X))
```

```
Z=INT(LØG(X+5))
```

3.5. L'instruction de branchement inconditionnel.

Nous avons vu que chaque ligne ne peut comporter qu'une seule instruction BASIC et doit comporter en outre systématiquement un numéro de ligne qui est utilisé comme ÉTIQUETTE.

L'instruction de branchement GO TO a l'allure suivante :

GO TO *n* ou GOTO *n*,

n étant un numéro d'étiquette.

Nous verrons en 3.9 une autre forme de l'instruction GO TO : L'instruction GOTO calculé.

3.6. L'instruction de test arithmétique élémentaire.

En mathématiques, les comparaisons se font surtout sur des nombres réels et se limitent à : plus petit, plus grand, égal, etc.

En BASIC, nous ferons appel à des opérateurs de comparaison qui sont donnés par le tableau suivant :

Opérateur	*Signification*
<	plus petit que
>	plus grand que
=	égal
< > (1)	différent de
< = (2)	plus petit ou égal
> = (2)	plus grand ou égal

FIG. 3.1.

Les instructions de comparaison sont construites à partir de ces opérateurs ; il faudra en outre, indiquer le numéro de l'instruction à exécuter dans le cas où la comparaison est satisfaite. La forme générale de l'instruction de comparaison élémentaire est :

IF ⟨expression arithmétique⟩ ⟨opérateur de comparaison⟩ ⟨expression arithmétique⟩ THEN ⟨numéro de ligne⟩

Exemple :

```
050 IF A>X THEN 100

500 IF A-X>=COS(X+3) THEN 300
```

Cette instruction permet d'écrire complètement la séquence BASIC relative à la résolution de l'équation du second degré.

```
0050 D=B↑2-4*A*C
0060 IF D<0 THEN 100
0070 IF D>0 THEN 120
0080 X1=X2=-B/(2*A)
0090 GO TO 140
0100 PRINT "PAS DE RACINE REELLE"
0110 STOP
0120 X1=(-B-SQR(D))/(2*A)
0130 X2=(-B+SQR(D))/(2*A)
0140 PRINT "X1=";X1,"X2=";X2
0150 END
```

3.7. Exercices.

1. Écrire en BASIC une séquence d'instructions permettant de calculer le PGCD de deux nombres A et B. Utiliser pour cela l'organigramme 1.3 et les fonctions standards.

(1) Certains systèmes acceptent en outre, le caractère ≠ pour désigner cet opérateur.

(2) Certains systèmes acceptent indifféremment les écritures = < et = > ou < = et > = .

On supposera pour le moment que A et B sont en mémoire, le PGCD sera rangé dans une variable appelée P.

2. Écrire la séquence permettant de calculer le nombre *e* à partir de la série $\frac{1}{n!}$ en utilisant les deux organigrammes du chapitre 1.

3. Faire l'organigramme et écrire le programme correspondant à la recherche de l'indice et de la valeur numérique du plus grand élément d'un tableau A(10) supposé être déjà lu.

1. Cas sans utiliser la fonction MAX.
2. Cas en utilisant la fonction MAX si on ne s'intéresse pas à l'indice de l'élément le plus grand.

Solutions :

1. *Calcul du PGCD*

```
20 LET Q=INT(A/B)
30 LET R=A-Q*B
40 IF R=0 THEN 80
50 LET A=B
60 LET B=R
70 GO TO 20
80 LET P=B
```

2. *Calcul de e* (*Premier cas*) :

```
0010 LET E=F=N=1
0020 LET F=F/N
0030 LET X=E+F
0040 IF X=E THEN 80
0050 LET E=X
0060 LET N=N+1
0070 GØ TØ 20
0080 PRINT N,E
0090 STØP
0100 END
```

Deuxième cas :

```
0010 DIM X(20)
0020 LET E=0
0030 LET X(1)=1
0040 LET I=2
0050 LET X(I)=X(I-1)/I
0060 LET I=I+1
0070 IF I<=20 THEN 50
0080 LET I=20
0090 LET E=E+X(I)
0100 LET I=I-1
0110 IF I>=1 THEN 90
0120 PRINT E+1
0130 END
```

ou

```
0080 LET I=I-1 (1)
0090 LET E=E+X (1)
0100 IF I>=1 THEN 80
0110 PRINT E+1
0120 END
```

(1) Quand on passe de l'instruction 70 à l'instruction 80 I vaut ici 21. La solution de gauche est cependant meilleure.

Nous verrons plus loin une méthode plus concise permettant de traiter cet exemple.

3. *Recherche du plus grand élément d'un tableau.*

Une méthode très simple consiste à utiliser une variable X qui contiendra la valeur du maximum et une autre variable J qui contiendra la valeur de l'indice correspondant au maximum.

On initialise X et J avec les valeurs respectives A(1) et 1, puis on compare successivement X avec A(2), A(3)... A(10).

Si $X \geqslant A(I)$ on fait progresser I : $(I = I + 1)$.

Si $X < A(I)$ on opère le changement suivant :

$$X = A(I)$$
$$J = I$$

puis on fait progresser I.

Finalement on obtient dans X et J les quantités désirées.

L'organigramme est donc le suivant :

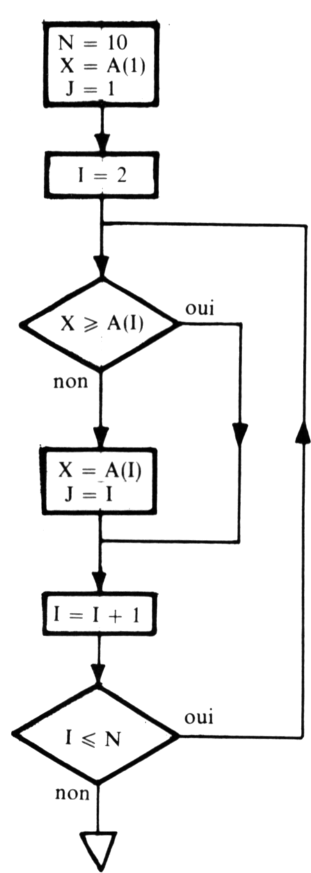


FIG. 3.2.

Le programme correspondant est donc le suivant :

Premier cas :

```
0010 DIM A(10)
0020 MAT READ A
0030 LET N=10
0040 LET X=A(1)
0050 LET J=1
0060 FØR I=2 TØ N
0070   IF X>=A(I) THEN 100
0080   LET X=A(I)
0090   LET J=I
0100 NEXT I
0110 PRINT J,X
0120 .......
0130 .......
```

Deuxième cas :

```
0010
0020
0030
0040
0050
0060 FØR I=2 TØ N
0070   LET X=MAX(X,A(I))
0080 NEXT I
```

3.8. Forme étendue de l'instruction IF.

Au BASIC originel, ont été adjoints trois opérateurs supplémentaires (1) permettant d'utiliser une forme plus évoluée de l'instruction IF. Ces opérateurs sont NOT, AND et OR.

Remarque : Effectuer une comparaison E1 > E2 par exemple, revient à calculer les valeurs numériques de E1 et E2 et de répondre ensuite :

oui E1 > E2 ou
non E1 n'est pas plus grand que E2

La première réponse est E1 > E2 vrai

La deuxième réponse est E1 > E2 faux.

Nous pouvons donc considérer le résultat d'une comparaison comme une variable « Booléenne » (ou « logique »). Le résultat de deux comparaisons successives pourra alors être repéré par la combinaison de deux variables booléennes relatives à chaque comparaison.

(1) Ce n'est pas le cas pour certains petits systèmes tels que : MULTI-8, VARIAN, BAMI.

Considérons l'organigramme suivant :

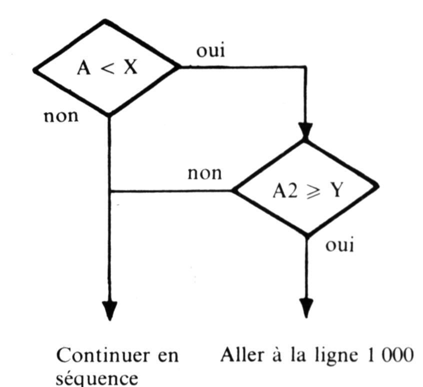


FIG. 3.3.

Avec l'instruction IF élémentaire, nous écrirons :

```
100 IF  A < X     THEN 110
105 GØ TØ 120
110 IF  A ↑ 2 > = Y THEN 1 000
120
```

ou encore

```
100 IF  A < = X  THEN 120
110 IF A ↑ 2 > = Y THEN 1 000
120
```

Ces deux séquences précédentes peu élégantes pourront être remplacées par l'instruction suivante beaucoup plus concise :

```
100   IF   A < X   AND   A ↑ 2 > = Y   THEN 1 000
120
```

La signification de ces trois opérateurs est la suivante :

NØT : opérateur unaire qui s'applique à l'expression qui la suit immédiatement, permet de s'intéresser au contraire de la comparaison.

IF A > = X ... peut s'écrire : IF NØT (A < X) ...

AND : opérateur binaire qui relie deux comparaisons. Le résultat est vrai si les deux comparaisons sont vraies.

ØR : opérateur binaire qui relie deux comparaisons. Le résultat est faux si les deux comparaisons ont donné pour résultat faux. Le résultat est vrai dans les autres cas.

Exemple :

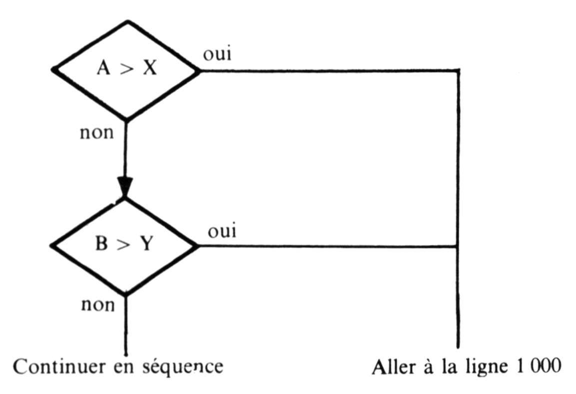


FIG. 3.4.

Cet organigramme se traduit par l'instruction suivante :

IF A > X ØR B > Y THEN 1 000

Certains systèmes disposent de formes plus évoluées de l'instruction IF qui sont signalées au chapitre 11.

3.9. *L'instruction GO TO calculée.*

Considérons la portion d'organigramme suivante :

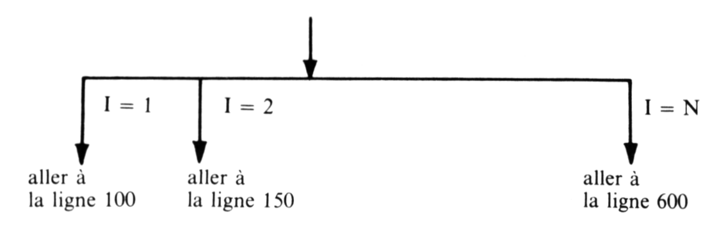


FIG. 3.5.

Ce test à sorties multiples peut se traduire à l'aide de ce que nous avons vu par la séquence suivante :

```
IF  I = 1  THEN  100
IF  I = 2  THEN  150

IF  I = N  THEN  600
```

Pour éviter cette séquence peu élégante, il a été ajouté au BASIC originel, l'instruction « GO TO calculée », similaire à celle du FORTRAN IV.

La forme la plus courante de cette instruction est :

ØN E GØ TØ $s_1, s_2, ..., s_n$

dans laquelle E est une expression arithmétique et $s_1, s_2, ..., s_n$ sont des numéros de lignes.

L'exemple ci-dessus s'écrirait :

ØN I GØ TØ 100, 150, ..., 600

REMARQUE :

1. Cette instruction existe sur certains systèmes sous une forme un peu différente :

CALL	360	GØ TØ	$s_1, s_2, \quad s_n$	ØN E
VARIAN	620	GØ TØ	$s_1, s_2, \quad s_n$	ØF E

mais la signification générale est la même.

2. Cette instruction n'existe pas sur les petits systèmes (exemple MULTI-8).

3.10. Exercices.

1) Représenter en BASIC la portion d'organigramme suivante :

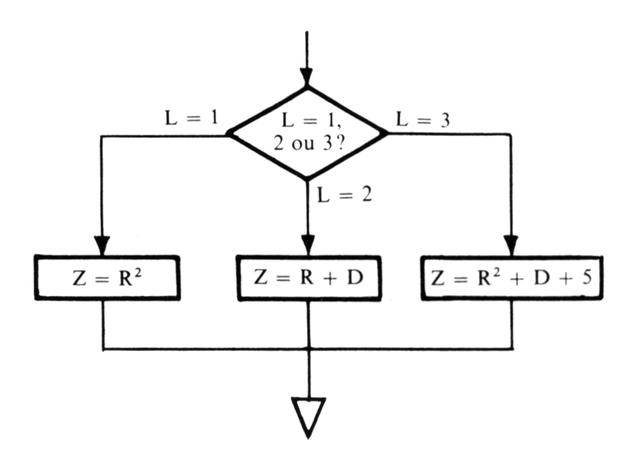


FIG. 3.6.

2) Représenter en BASIC la portion d'organigramme suivante, la variable L ne pouvant prendre que les valeurs 1, 2, 3 ou 4.

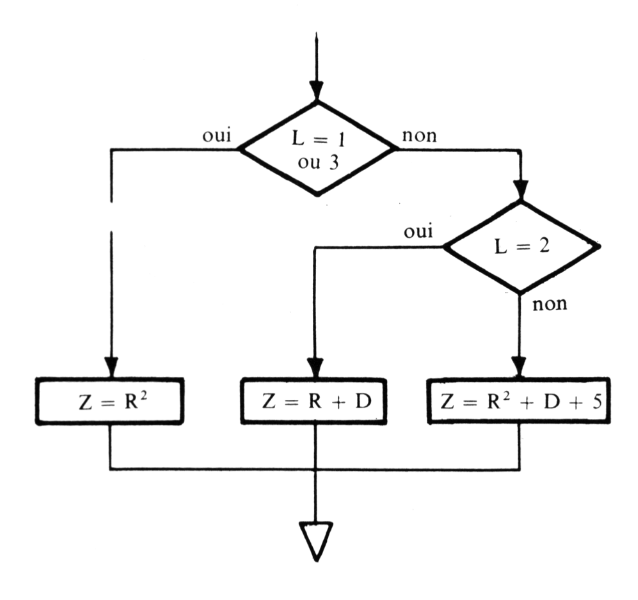


FIG. 3.7.

Réponses :

1. Sans utilisation du GOTO calculé.

```
0100 IF L=1 THEN 140
0110 IF L=2 THEN 160
0120 LET Z=R*R+D+5
0130 GØ TØ 170
0140 LET Z=R*R
0150 GØ TØ 170
0160 LET Z=R+D
0170 .........
```

— Avec utilisation du GOTO calculé.

```
0100 ØN L GØ TØ 110,130,150
0110 LET Z=R*R
0120 GØ TØ 160
0130 LET Z=R+D
0140 GØ TØ 160
0150 LET Z=R*R+D+5
0160 ..............
0170
```

2. Avec utilisation du IF et de l'opérateur OR.

```
0100 IF L=1 ØR L=3 THEN 140
0110 IF L=2 THEN 160
0120 LET Z=R*R+D+5
0130 GØ TØ 170
0140 LET Z=R*R
0150 GØ TØ 170
0160 LET Z=R+D
0170 .........
```

— Avec utilisation du GOTO calculé :

```
0100 ØN L GØ TØ 110,130,150
0110 LET Z=R*R
0120 GØ TØ 160
0130 LET Z=R+D
0140 GØ TØ 160
0150 LET Z=R*R+D+5
0160 .............
```

3.11. Les instructions de contrôle : PAUSE, STOP, END.

Ces instructions permettent toutes de suspendre l'exécution d'un programme. Leur forme générale se compose du numéro de ligne suivi du « mot clé » PAUSE, STOP ou END :

100 PAUSE

....

500 STØP

....

600 END

Elles provoquent, lors de l'exécution du programme, l'impression d'un message sur la console utilisateur. Ce message indique le type d'instruction rencontré et son numéro de ligne :

Exemple : STØP AT LINE 500
PAUSE AT LINE 100

Les instructions PAUSE et STOP peuvent figurer plusieurs fois au sein d'un même programme. Par contre, l'instruction END ne peut figurer qu'une seule fois et seulement en fin de programme.

3.11.1. L'instruction PAUSE :

Cette instruction suspend l'exécution du programme et redonne l'initiative à l'utilisateur qui peut alors demander :

— ou bien de reprendre l'exécution au point où elle avait été interrompue ;

— ou bien d'abandonner cette exécution afin de passer à autre chose (modification de programme, nouveau programme, arrêt du travail, etc.).

Cette instruction présente un intérêt en cours de mise au point et en cours d'exécution de programmes longs.

En cours de mise au point en plaçant aux endroits importants des ordres PAUSE, l'utilisateur peut suivre le cheminement du programme et mieux localiser les « erreurs de logique ».

En cours d'exécution, avant d'attaquer par exemple une partie nécessitant un temps de calcul important on peut placer un ordre PAUSE. Ainsi

au vu des résultats intermédiaires on peut décider d'abandonner la suite de l'exécution ou au contraire la poursuivre. Cette méthode permet sur les systèmes Time Sharing d'économiser du temps de calcul.

3.11.2. L'instruction STOP :

Cette instruction provoque l'abandon du programme sans possibilité de reprise.

Exemple :

```
050 IF A<>0 THEN 70
060 STØP
070 IF B↑2-4*A*C<0 THEN 60
080 X1=......
090 X2=......
.............
.............
120 STØP
.............
```

3.11.3. L'instruction END :

Cette instruction est destinée à indiquer au système que le programme est terminé. Cette instruction doit donc figurer *une fois et une seule* dans un programme et doit être *située à la dernière ligne* du programme.

Toutefois, certains systèmes acceptent que l'utilisateur omette cette instruction.

3.12. L'instruction REM.

L'instruction REM permet d'insérer un texte dans le programme, texte qui n'est pas analysé par le compilateur mais est destiné à aider l'utilisateur à se rappeler ce qu'il a voulu faire dans son programme.

Exemple :

```
0100 REM CALCUL DE LA RACINE
0110 LET X=X0
0120
0130
     ......
0200 REM VERIFICATIØN
0210
     ......
0250 REM IMPRESSIØN DES RESULTATS
0260 PRINT
0270 FØR I=1 TØ N
0280  PRINT A(I)
0290 NEXT I
0300
     ......
0410 END
```

CHAPITRE 4

LES BOUCLES DE CALCUL

4.1. Introduction.

Dans de nombreux problèmes il est nécessaire de répéter plusieurs fois une série d'instructions. Considérons par exemple les deux problèmes suivants :

— multiplication des éléments d'un tableau par un nombre donné,

— calcul de la somme des n premiers termes de la série

$$\frac{1}{3} + \frac{1}{5} + \frac{1}{7} +$$

pour lesquels l'indice va de 3 à $2n + 1$.

Les organigrammes correspondants sont les suivants :

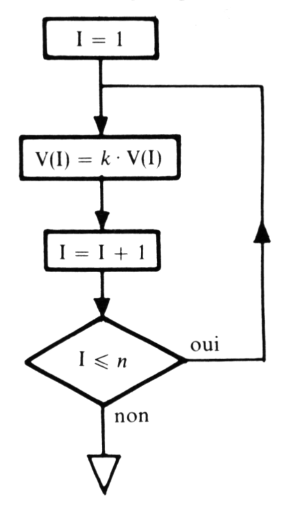


FIG. 4.1.

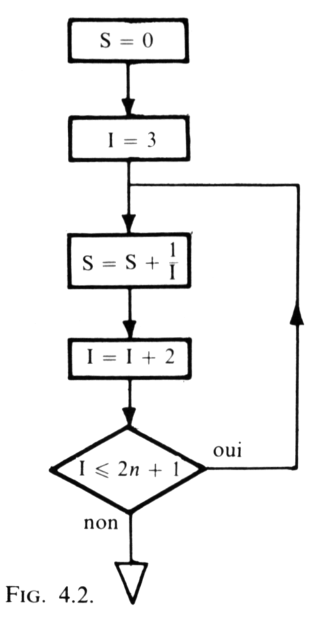


FIG. 4.2.

Le schéma général d'une telle boucle de calcul est :

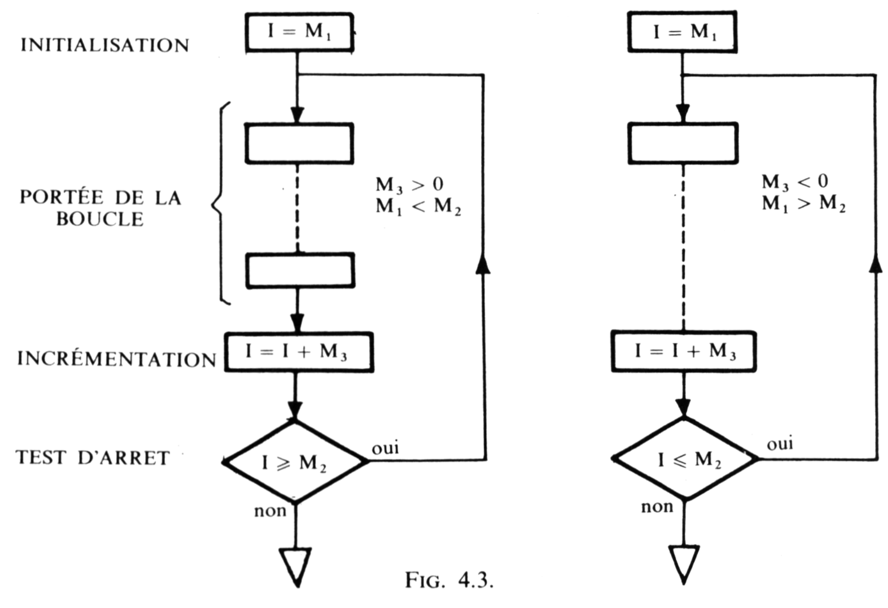


FIG. 4.3.

M1, M2, M3 représentent respectivement

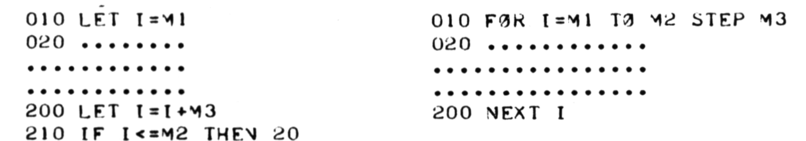
— la valeur initiale,

— la valeur finale,

— le pas d'incrémentation de la variable de contrôle I.

4.2. *Forme générale des instructions FOR et NEXT.*

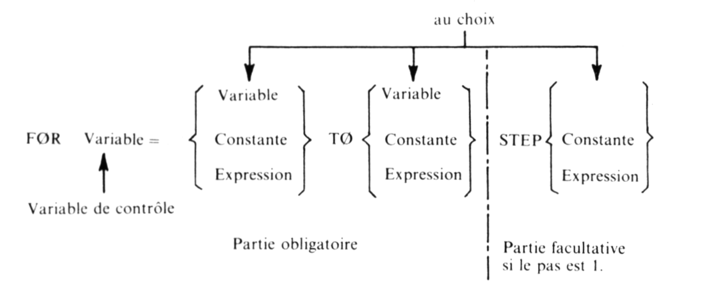
L'organigramme précédent pourra se traiter de la façon suivante :



```
010 LET I=M1
020 ........
...........
...........
200 LET I=I+M3
210 IF I<=M2 THEN 20
```

```
010 FØR I=M1 TØ M2 STEP M3
020 ............
...............
...............
200 NEXT I
```

La forme générale de l'instruction FOR est la suivante



REMARQUE : Le système XDS-940 accepte des formes légèrement différentes mais ayant la même signification.

```
FØR I=E1 STEP E2 TØ E3
FØR I=E1 TØ E3 BY E2
```

Exemple :

```
FØR X=1 TØ J+1 STEP H
```

La forme générale de l'instruction NEXT est

NEXT ⟨ variable ⟩.

La variable étant celle correspondant à la variable de contrôle de l'instruction FOR associée.

Exemple :

```
FØR I=1 TØ 10
.............
NEXT I
```

correct

```
FØR I=1 TØ 10
.............
NEXT J
```

incorrect

4.3. Exemples simples d'utilisation.

La forme très souple des instructions NEXT et FOR permet de décrire de façon concise certains calculs :

Exemple 1 : calcul des dix premiers termes de la série

$$\frac{1}{1}+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}+\ldots+\frac{1}{19}.$$

```
010 LET S=0
020 FØR I=1 TØ 19 STEP 2
030 LET S=S+1/I
040 NEXT I
```

Exemple 2 : calcul des dix premiers termes de la série précédente en effectuant les calculs dans l'ordre inverse.

```
010 LET S=Ø
020 FØR I=19 TØ 1 STEP -2
030 LET S=S+1/I
040 NEXT I
```

4.4. Règles élémentaires relatives à une boucle de calcul.

— L'instruction NEXT doit être postérieure au FOR associé.

— Si les paramètres M_1, M_2, M_3 sont des variables ou des expressions

arithmétiques, *alors leur valeur numérique est calculée avant l'entrée dans la portée de l'instruction FOR.*

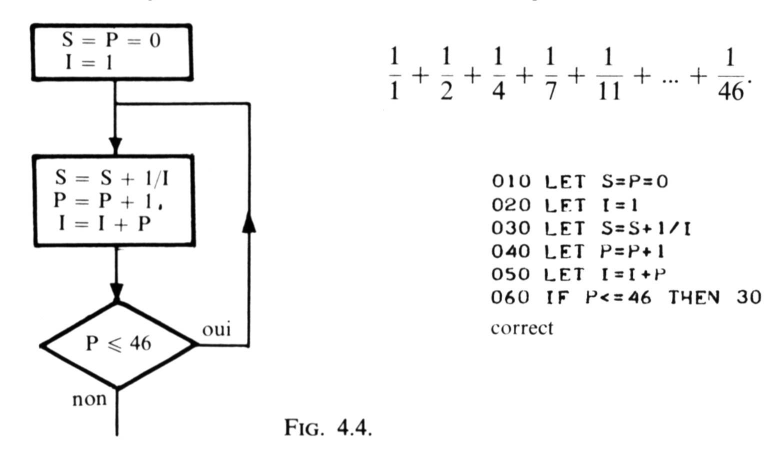
Remarques :

1. Certaines règles sont dictées par des considérations de logique et d'autres par des contraintes dues à la réalisation des compilateurs. Par exemple, la règle relative à NEXT est dictée par des considérations de logique.

Certaines restrictions sur la souplesse du langage facilitent la réalisation de compilateurs (réalisation qui pose des problèmes techniques) et permettent parfois de meilleures performances à l'exécution.

2. Certains systèmes acceptent des boucles « dynamiques » (cf. chap. XI).

Exemple : calcul de la somme des dix premiers termes de la série :



$$\frac{1}{1}+\frac{1}{2}+\frac{1}{4}+\frac{1}{7}+\frac{1}{11}+\ldots+\frac{1}{46}.$$

```
010 LET S=P=0
020 LET I=1
030 LET S=S+1/I
040 LET P=P+1
050 LET I=I+P
060 IF P<=46 THEN 30
```

correct

Fig. 4.4.

Ceci donne un résultat faux.

```
010 LET S=P=0
020 FØR I=1 TØ 46 STEP P+1
030 LET S=S+1/I
040 NEXT I
```

En effet la valeur P + 1 est calculée avant l'entrée dans la portée (ici l'instruction 30) et la valeur du Pas reste 1 au lieu de croître.

La séquence suivante est également incorrecte.

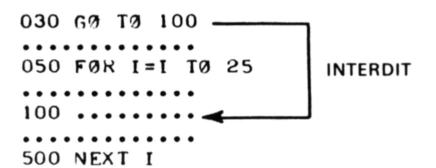
```
010 LET S=P=0
020 FØR I=1 TØ 46 STEP P
030 LET S=S+1/I
040 LET P=P+1
050 NEXT I
```

En effet, la valeur de P qui est prise pour pas de calcul est 0 alors que P est incrémenté ensuite régulièrement. On peut donc tourner éternellement dans la boucle FOR.

— Si le pas est positif, les itérations sont stoppées dès que la variable de contrôle devient supérieure à M2.

Si le pas est négatif, les itérations sont stoppées dès que la variable de contrôle devient inférieure à M2.

— Il est interdit de sauter à l'intérieur de la portée d'une boucle sans exécuter l'instruction FOR d'initialisation.



Toutefois ceci n'est pas systématiquement détecté par les compilateurs et dans ce cas l'interprétation à l'exécution peut varier d'un ordinateur à l'autre.

— On peut quitter une boucle FOR avant que la variable de contrôle ait atteint la valeur limite.

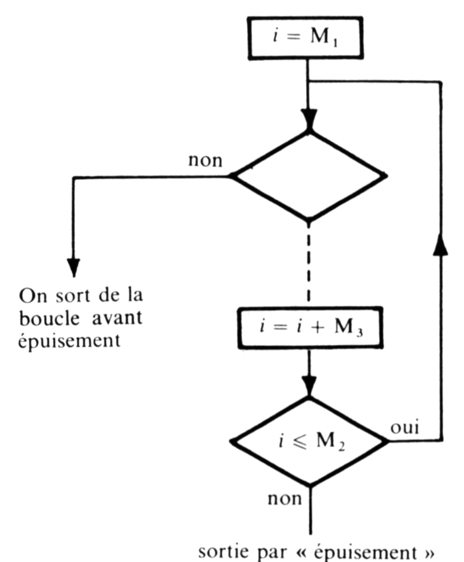


FIG. 4.5.

4.5. *Exemple d'utilisation.*

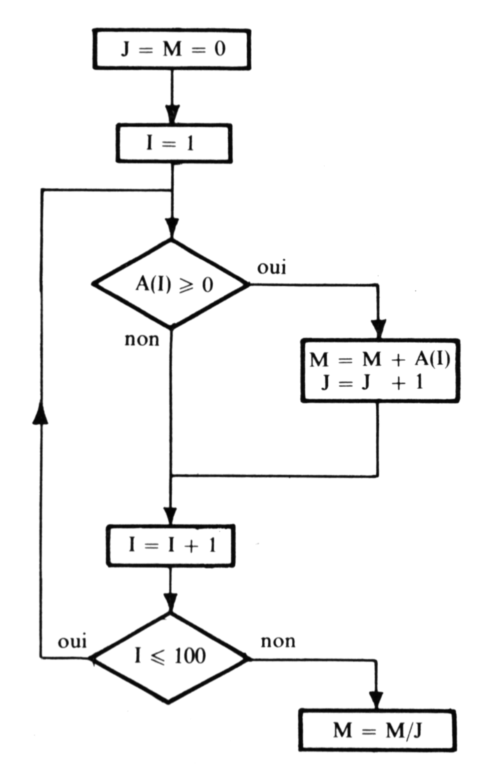
Les exemples suivants sont destinés à montrer la facilité et la souplesse d'utilisation des instructions FOR et NEXT ;

— calcul d'une moyenne : soit un tableau de 100 nombres dont les valeurs sont en mémoire : calculer la moyenne M ;

```
100 DIM A(100)
110 LET M=0
120 FØR I=1 TØ 100
130 LET M=M+A(I)
140 NEXT I
150 LET M=M/100
```

— même problème mais on ne calcule la moyenne que pour les éléments positifs ou nuls du tableau.

Il faut donc compter le nombre d'éléments positifs.




```
100 LET J=M=0
110 FØR I=1 TØ 100
120 IF A(I)<0 THEN 150
130 LET M=M+A(I)
140 LET J=J+1
150 NEXT I
160 LET M=M/J
```

FIG. 4.6.

— Calcul de la somme des termes de la série

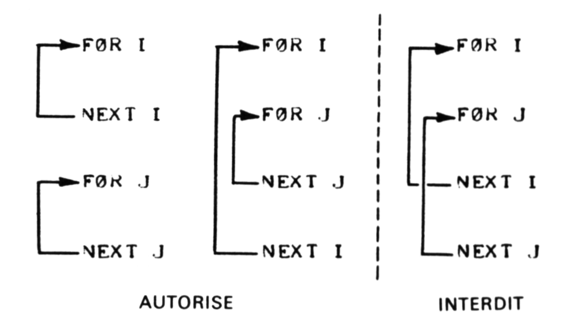
$$1 + \frac{1}{2} - \frac{1}{3} + \frac{1}{4} - \frac{1}{5} + \frac{1}{6} \cdots$$

On arrête le calcul dès que le dénominateur atteint 10 000 en valeur absolue :

```
LET S=1
FØR I=2 TØ 10000 STEP 2
LET S=S+1/I-1/(I+1)
NEXT I
```

4.6. Règles relatives à plusieurs boucles de calcul.

Un programme peut contenir plusieurs boucles de calcul. Ces boucles peuvent être « séquentielles » ou emboîtées *mais ne doivent pas se chevaucher.*



4.7. Valeur de la variable de contrôle à la sortie d'une boucle FOR.

Lorsque l'on quitte une boucle FOR avant que la valeur de la variable de contrôle ait atteint la valeur limite, la valeur de la variable de contrôle est égale à celle qu'elle avait juste avant de quitter la boucle.

Exemple :

```
100 FØR I=1 TØ 100
.................
150 IF A(I)<B(I) THEN 500
.................
200 NEXT I
.................
500 ..............
```

Si le test A(I) < B(I) est satisfait lorsque I atteint par exemple la valeur 48, il y aura saut à la ligne 500 et la valeur de I en 500 sera toujours 48.

Par contre lorsqu'on quitte une boucle FOR par épuisement, la valeur de la variable de contrôle I peut être *quelconque*. Cependant pour de nombreux systèmes, sa valeur correspond à la dernière valeur possible augmentée de la valeur du pas.

Exemple :

```
100 FØR I=1 TØ 10 STEP 2
.................
200 NEXT I
210 ..............
```

I prend les valeurs successives 1, 3, 5, 7, 9,

la valeur de I lorsque l'on passe de l'instruction 200 à 210 sera 11.

Ceci est dû au fait que l'on exécute d'abord l'instruction de progression I = I + PAS et que l'on compare ensuite I avec la valeur finale.

Conseil : pour améliorer la lisibilité d'un programme, on peut décaler certaines instructions vers la droite notamment les instructions de la portée d'une instruction FOR (cf. exemple en 4.8.1).

4.8. Exercices.

Programmer le calcul de π à partir des séries $\frac{\pi^2}{6}$, $\frac{\pi^2}{8}$, $\frac{\pi^2}{12}$ et $\frac{\pi^4}{90}$ en utilisant les organigrammes donnés en 2.5.

4.8.1. Calcul de π à partir de la série $\frac{\pi^2}{6}$.

```
0010 LET S=1
0020 FØR I=2 TØ 40000
0030   LET X=S+1/(I*I)
0040   IF X=S THEN 80
0050   LET S=X
0060 NEXT I
0070 LET I=40000
0080 PRINT "I=";I;"        PI=";SQR(6*S)
0090 END
```

Cette instruction s'explique par le paragraphe 4.7 ici en toute rigueur on quittera la boucle bien avant que i n'atteigne 40 000 et on peut omettre cette instruction sans courir le risque d'imprimer $i = 40\,001$ au lieu de 40 000 !

4.8.2. Calcul de π à partir des séries $\frac{\pi^2}{8}$, $\frac{\pi^2}{12}$ et $\frac{\pi^4}{90}$.

Par rapport au programme précédent il suffit de modifier deux ou trois lignes :

$\frac{\pi^2}{8}$

```
0020 FØR I=3 TØ 40000 STEP 2
...............
0070 LET P=SQR(8*S)
```

$\frac{\pi^2}{12}$

```
0010 LET S=0
0020 FØR I=1 TØ 20000 STEP 2
0030 LET X=S+1/I↑2 -1/(I+1)↑2
0040 IF X=S THEN 80
0050 LET S=X
0060 NEXT I
0070 LET I=20000
0080 PRINT "I=";I;"PI=";SQR(12*S)
0090 END
```

$\frac{\pi^4}{90}$

```
0010 LET S=0
0015 FØR I=1 TØ 10000
0020 LET X=S+1/I↑4
0025 IF X=S THEN 50
0030 LET S=X
0035 NEXT I
0040 LET I=10000
0050 PRINT "I=";I;"        PI=";SQR(SQR(90*X))
0060 END
```

4.8.3. Refaire l'exercice n° 3 du paragraphe 3.7 en utilisant les instructions FOR et NEXT.

```
0010 DIM A(10)
0015 MAT READ A
0020 LET N=10
0030 LET X=A(1)
0040 LET J=1
0050 FØR I=2 TØ N
0060 IF X>=A(I) THEN 90
0070 LET X=A(I)
0080 LET J=I
0090 NEXT I
0110 PRINT "XMAX="; X; " CØRRESPØNDANT A L'INDICE"; J;
```

Partie correspondant à l'organigramme. (lignes 0030 à 0090)

ANNEXE : Influence des erreurs d'arrondi sur la précision des résultats.

Pour calculer la somme des séries, on peut faire le calcul dans l'ordre habituel ou à l'envers,

Exemple : Pour $\frac{\pi^2}{8}$ LET S = 0
FØR I = 40 001 TØ 3 STEP - 2
....
....

On constate qu'en opérant à l'envers, le programme est un peu plus difficile à écrire, mais donne des résultats plus précis, car les erreurs d'arrondi jouent un rôle plus faible. Avec un ordinateur dont la mantisse est de 31 bits, nous avons obtenu les résultats suivants :

SÉRIES / TYPE SÉRIE	*Série normale*		*Série inverse*	
	Nombre de termes	Valeur obtenue	Nombre de termes	Valeur obtenue
Série $\frac{\pi^2}{6}$	10 000	3.14149	10 000	3.1414972
			20 000	3.1415449
	32 769 (1)	3.1415677	32 769	3.1415635
Série $\frac{\pi^2}{8}$			10 001	3.1415290
			20 001	3.1415608
	32 769 (1)	3.1415760	32 769	3.1415732
			40 001	3.1415767
Série $\frac{\pi^2}{12}$	1 625 (1)	3.14159235	40 001	3.1415927
Série $\frac{\pi^4}{90}$	182 (1)	3.14159265	10 000	3.14159265

(1) A partir de ces valeurs la précision n'augmente plus, avec le nombre de termes. Ce tableau montre en outre que certaines séries convergent beaucoup plus vite que d'autres et sont moins sensibles aux erreurs d'arrondi. La valeur à obtenir commence par 314159265358979.

Les 9 premiers chiffres sont obtenus avec 182 termes par la série $\frac{\pi^4}{90}$.

CHAPITRE 5

INSTRUCTIONS D'ENTRÉES/SORTIES

5.0. Introduction.

Lors de l'exécution d'un programme, le besoin d'échanges d'informations entre l'ordinateur et utilisateur est satisfait par l'intermédiaire d'instructions spéciales appelées : « Instructions d'entrées/sorties » :

— entrée : sens utilisateur vers ordinateur,

— sortie : sens inverse.

Le BASIC comporte plusieurs types d'instructions d'entrées/sorties :

— instructions élémentaires,

— instructions d'entrées/sorties de tableau,

— instructions avec image.

En outre, des instructions pour le « traitement de fichiers » ont été ajoutées au BASIC (cf. chap. 10).

Dans le détail, la forme exacte des instructions d'entrées/sorties varie d'un constructeur à l'autre. Nous indiquerons donc seulement l'allure générale de ces instructions.

Les « périphériques » d'entrées/sorties disponibles en BASIC sont limitées souvent à une machine à écrire (le terminal de l'utilisateur). Parfois, l'utilisateur peut disposer en plus d'une imprimante rapide. Les fichiers peuvent être stockés sur disque magnétique.

5.1. Généralités sur les instructions d'entrées/sorties élémentaires.

Ces instructions comportent l'un des mots clés suivants :

READ
ou } pour les instructions d'entrée
INPUT

PRINT pour l'instruction de sortie

5.2. Les deux types d'instructions d'entrées.

Une opération d'entrée de données peut être faite :

— à partir de données préparées avant le début de l'exécution du programme et rangées dans des lignes numérotées (données statiques) ; on utilisera alors l'instruction READ,

— à partir de données que l'on introduit en cours d'exécution du programme (données dynamiques). Dans ce cas, on devra utiliser l'instruction INPUT.

5.3. Forme des instructions d'entrées.

READ liste de variables simples ou d'éléments de tableaux

INPUT "constante caractère" liste de variables simples ou d'éléments de tableaux
la constante caractère est optionnelle.

Exemples

```
010 DIM     X(10)
000
100 READ A, B, X(I), Z
110 INPUT « DONNER PRIX UNITAIRE », P
```

Une liste est une suite d'identificateurs séparés les uns des autres par une virgule.

REMARQUE : Les instructions READ et INPUT ne permettent pas de lire directement un tableau complet : il faut faire appel à une boucle. Nous verrons plus loin une autre méthode qui consiste à faire appel aux instructions spéciales pour tableaux (dites instructions matricielles).

Les nombres lus sont alors affectés aux variables selon l'ordre de la liste. Mais les nombres à lire ne doivent pas se présenter de n'importe quelle manière.

5.3.1. Instruction INPUT

Cette instruction provoque l'impression de l'éventuelle constante caractère placée juste après le mot clé INPUT, suspend l'exécution du programme afin que l'utilisation puisse introduire ses données. L'utilisateur est en général prévenu que son programme attend des données par l'envoi d'un caractère particulier (point d'interrogation, coup de sonnette). Les données doivent être introduites et séparées les unes des autres par une virgule ; lorsque toutes les données sont introduites, l'utilisateur envoie un « retour chariot ».

L'ordinateur vérifie alors si la liste des données permet de satisfaire la liste d'entrée. Si oui, il poursuit l'exécution du programme. Sinon, il émet soit un message d'erreur s'il y a une faute dans la liste (lettre au lieu de chiffre — absence de séparateurs — par exemple) ou une nouvelle demande d'entrée si les données précédemment envoyées n'étaient pas assez nombreuses pour satisfaire la liste.

Exemple :

```
0010 INPUT "DONNER LE PRIX UNITAIRE ",P
0020 PRINT
0030 PRINT "PRIX UNITAIRE = ";P
```

Résultat à l'exécution :

```
DONNER LE PRIX UNITAIRE 120.55

PRIX UNITAIRE =  120.55
```

REMARQUE : Cette méthode très souple pour l'utilisateur est largement employée par les sociétés de service Time Sharing pour leurs programmes d'application, car ceci pousse l'utilisateur à consommer de façon abusive du temps de connexion et du temps d'unité centrale. Elle s'avère cependant peu commode lorsqu'il faut entrer un grand nombre de données. Dans ce cas on préfère souvent utiliser l'instruction READ.

5.3.2. Instruction READ et DATA

Sur la plupart des systèmes, ces données devront être écrites sur des lignes numérotées qui accompagnent le programme proprement dit et qui contiennent le mot clé DATA.

Exemple :

```
..................
100 READ A,B,C,X
..................
500 DATA 10,-20.1,50
510 DATA 0
..................
```

Ici, lorsque l'on exécutera l'instruction READ, A prendra pour valeur 10, et B, C et X, respectivement les valeurs – 20.1, 50 et 0.

Les nombres doivent être séparés les uns des autres par un « séparateur ». Sur la plupart des systèmes, ce séparateur doit être une virgule. Sur certains systèmes, deux blancs ou une lettre peuvent constituer un séparateur.

Si une ligne DATA ne contient pas suffisamment de nombres l'ordinateur, pour satisfaire la liste des variables, fera appel aux nombres trouvés dans la prochaine instruction DATA et ainsi de suite jusqu'à ce que la liste des variables soit complètement satisfaite.

Si le nombre de données est insuffisant pour satisfaire la liste, le système abandonne en général l'exécution du programme et provoque l'impression d'un diagnostic d'erreur.

5.3.3. L'instruction RESTORE.

Cette instruction permet de relire les données lues à partir de l'instruction DATA à partir du début :

Exemple :

```
.........
100 READ A,B,C
110 READ P1,E
.........
150 RESTØRE
.........
200 READ X,Y,Z,W
.........
500 DATA 3,4,5,3.14159
510 DATA 2.714,8,10,15
```

L'instruction du numéro 100 fera attribuer à A, B, C, les valeurs 3, 4 et 5. L'instruction 110 attribuera à P1 et E, les valeurs 3.14159 et 2.714.

L'exécution de l'instruction RESTORE a la conséquence suivante : lors de la prochaine instruction READ (ici 200), au lieu de poursuivre la lecture en séquence, on exécutera la lecture en reprenant les données à leur début.

Ici, on attribuera donc aux variables X, Y, Z, W, respectivement les valeurs 3, 4, 5 et 3.14159.

REMARQUE : Certains BASIC (celui de MICROSOFT par exemple) acceptent la forme

RESTORE *n*

n étant un numéro de ligne.

Cette forme permet de relire des données non pas à partir de la première instruction DATA, mais à partir de l'instruction DATA indiquée par le numéro de ligne *n*.

5.4. Instruction de sortie PRINT.

La forme générale de cette instruction est :

PRINT ⟨ liste ⟩.

Mais ici, une liste a une forme plus générale que pour les instructions d'entrée : elle peut contenir non seulement

— des identificateurs de variable simple,

— des éléments de tableau,

mais en plus, des EXPRESSIONS ARITHMÉTIQUES et des ÉLÉMENTS DE MISE EN PAGE, qui sont :

— les séparateurs : — virgule,

— point virgule,

— des chaînes de caractères entre guillemets,

— la fonction TAB (qui sert à effectuer des tabulations),

— la barre oblique/(en anglais : SLASH) pour les systèmes PHILIPS, seulement.

Exemple *:*

PRINT " A = " ; A ; " B = " ; 3 * EXP(X/2) ; TAB (60), V(I).

Cette complexité s'explique par le fait que pour l'entrée, seul l'utilisateur est responsable de la présentation des données (présentation d'ailleurs souvent sans importance) alors que pour la sortie, le programme doit contenir les informations permettant à l'ordinateur d'effectuer la présentation désirée par l'utilisateur.

La présentation d'un nombre dépend des caractères de mise en page pour ce qui est de la longueur de la zone attribuée et de sa valeur numérique pour ce qui est de sa forme.

D'une manière générale, en BASIC, un nombre est édité de la façon suivante :

— forme entière si sa valeur numérique est entière et inférieure en valeur absolue à 1 000 000 000,

— forme décimale si sa valeur numérique n'est pas entière, mais reste compatible avec la longueur et la zone disponible,

— forme avec exposant dans les autres cas.

Dans le détail, la forme varie selon le système utilisé.

Forme PHILIPS 9200 :

— forme avec exposant si le nombre n'est pas entier ou si sa valeur absolue est supérieure à 999 999 999,

— le signe, si le nombre est négatif, est imprimé à gauche de la zone et la valeur numérique est cadrée à droite (ceci diffère des autres systèmes).

Forme SIGMA 5/7 :

— forme décimale pour nombres décimaux supérieurs en valeur absolue à 0,1 et inférieurs à 10^6 ou 10^{16} (selon le cas),

— forme avec exposant dans les autres cas.

Forme CDC 6000 :

— forme décimale pour les nombres non entiers lorsque l'édition n'est possible qu'avec 7 caractères (6 chiffres significatifs plus point décimal),

— forme avec exposant dans les autres cas : *n. nnnnn* E ± *nn*.

Forme système MARK II :

Pour tout nombre décimal, le système édite au plus 6 chiffres significatifs. Si la valeur est en valeur absolue, inférieure à 0,1 la forme avec exposant est utilisée sauf si la valeur peut être éditée sous forme décimale sans arrondi. Ainsi l'édition de .03456 correspond au nombre .0345600000 alors que 3.45600E-2 correspond à un nombre qui a dû être arrondi à .03456 pour l'édition.

La mise en page des chaînes de caractères est exposée au chapitre VIII.

5.4.1. Rôle de la virgule et du point virgule.

En BASIC, une ligne est artificiellement divisée en quatre zones dont la longueur est de l'ordre de 15 caractères (voir le tableau suivant).

Chaque variable sera éditée selon un format qui dépend notamment des facteurs suivants :

— valeur numérique du nombre (s'il est très grand ou très petit, il faudra faire appel à une forme avec exposant),

— séparateur qui suit la variable dans la liste : virgule ou point virgule,

— ordinateur utilisé.

Le tableau suivant donne une idée de ce qui est proposé par les constructeurs d'ordinateurs :

Ordinateurs / *Caractéristiques*	*IRIS 80*	*PHILIPS-HEWLETT-PACKARD-CDC-XDS-940*	*IBM*
Longueur de chaque zone	14	15	18 (2)
Nombre de zones par ligne	5	5	6 ou 7
Nombre de caractères par ligne	72 (1)	72 (1)	120 (1)

(1) La plupart des systèmes BASIC sont construits pour être utilisés à partir de terminaux du type ASR 33, d'où la longueur de 72 caractères. La longueur de la dernière zone est de 12 caractères pour de nombreux systèmes, et de 16 pour XDS sigma 5/7.

Par contre, IBM qui prône l'utilisation de machines à écrire à boules offre des lignes de 120 caractères, ce qui explique la plus grande longueur de zone.

(2) Cette longueur s'explique par la plus grande longueur de ligne et par le fait que le BASIC du CALL 360 offre la possibilité de travailler en « double précision ».

D'une manière générale, chaque valeur numérique est en BASIC cadrée à gauche de la zone qui lui est attribuée, le premier caractère étant réservé au signe (blanc si le nombre est positif).

Rôle de la virgule : Lorsqu'un nombre est suivi dans la liste par une virgule, il est édité en utilisant une zone complète. Ainsi, en utilisant des lignes de 72 caractères, on ne peut éditer que cinq nombres au maximum par ligne.

Exemple :

```
0100 LET A=3.24159
0110 LET B=-274.142
0190 PRINT "123456789012345678901234567890"
0200 PRINT A,-274.142
0210 PRINT A,B
0220 PRINTA;B
```

Nous aurons l'édition suivante (pour des zones de 15 car.). Le premier caractère de chaque zone est réservé au signe. Lorsque le nombre est positif, le signe + est omis et est remplacé par un blanc.

```
123456789012345678901234567890
 .324159E 01    -.274142E 03
 .324159E 01    -.274142E 03
 .324159E 01 -.274142E 03

STOP AT LINE  0220
```

Rôle du point virgule : Lorsqu'un nombre est suivi dans la liste par un point virgule, il est édité en utilisant une zone réduite. La longueur de cette zone dépend alors :

— de la valeur numérique du nombre,

— de l'ordinateur utilisé.

La forme de l'édition étant dans ce cas très variable, nous indiquerons quelques formes rencontrées sur différents systèmes :

Forme PHILIPS 9200 :

si la valeur numérique du nombre est entière et inférieure en valeur absolue à 10 000, le nombre est édité sur une zone de 6 caractères,

si la valeur numérique est plus grande en valeur absolue ou si elle n'est pas entière, le nombre est édité sous la forme avec exposant sur une zone de 13 caractères.

Forme IRIS 80 : Il n'y a pas de zone réduite de longueur fixe. Chaque zone a une longueur paire et dépend de la valeur numérique du nombre à imprimer. Chaque zone est donc réduite à la plus petite longueur paire compatible avec la valeur numérique du nombre à imprimer.

Forme IBM : Ce système distingue cinq longueurs « condensées » dont les valeurs sont : 6, 9, 12, 15 et 18.

L'impression débute à la position actuelle de la tête d'impression et se termine 6, 9, 12, 15 ou 18 caractères plus loin selon la valeur numérique du nombre.

Forme CDC CYBER : La zone est réduite à 6 caractères pour les entiers de 1 à 3 caractères, 9 caractères pour les entiers de 4 à 6 caractères, 12 caractères pour les nombres nécessitant 7 à 9 caractères. Pour les nombres plus grands une zone normale de 15 caractères leur est attribuée.

5.4.2. L'instruction de sortie avec fonction TAB.

Les méthodes précédentes permettent d'éditer nombres et chaînes de caractères avec peu de souplesse de mise en page. La fonction TAB permet dans une certaine mesure, de combler cette lacune.

TAB(N) peut apparaître dans une instruction PRINT et indique alors que la tête d'écriture du terminal doit se positionner au n^{eme} caractère de la ligne.

Exemple : PRINT A,TAB(30),B

Signifie que après l'impression de la valeur de A, il faudra positionner la tête d'impression devant le 30e caractère de la ligne, avant de commencer l'édition de B.

Si N n'est pas entier, TAB(N) est interprété de la même façon que TAB(INT(N)) c'est-à-dire que la tabulation se fait à partir de entier (N).

Si la tête d'écriture a dépassé la N^{eme} position, la fonction TAB(N) est sans effet car la tête ne recule pas.

Cette fonction de tabulation présente trois avantages :

— améliorer les possibilités de présentation des résultats,

— faciliter l'édition de tableaux (cf. les exercices),

— permettre de tracer des courbes sur le terminal, d'une façon relativement simple (cf. exercice 5.6.1).

5.4.3. Passage d'une ligne à la suivante en sortie.

— Lorsqu'une liste de variables est trop importante pour pouvoir être éditée sur une seule ligne, elle sera éditée sur plusieurs lignes.

— Lorsqu'une liste se termine par un séparateur (virgule ou point virgule), aucun « retour chariot » n'est effectué.

Cependant l'exécution de la prochaine instruction PRINT provoque :

- un passage à la ligne suivante s'il ne reste pas assez de place pour continuer sur cette ligne (la notion de place restante dépend de ce qu'il faut éditer),

- le début de l'impression sur la même ligne s'il reste de la place pour la ou les premières valeurs numériques ou chaînes de caractères à éditer.

Exemple 1 :

```
0010 PRINT 1,2,3,4,5,6,7,8,9,
0020 PRINT 10,11
0030 STØP
0040 END
```

Résultat de l'exécution

```
  1        2        3        4        5
  6        7        8        9       10
 11
```

Exemple 2 :

```
0010 PRINT 1;2;3;4;5;6;7;8;9;
0020 PRINT 10;11;12;13;14;
0030 STØP
0040 END
```

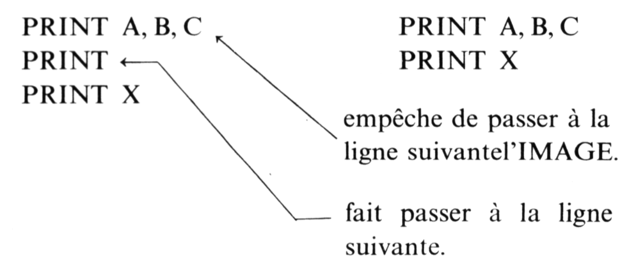
Résultat de l'exécution

```
 1    2    3    4    5    6    7    8    9    10    11    12
13   14
```

— Lorsqu'une liste de sortie ne se termine pas par un séparateur, un « retour chariot » est effectué après impression et la prochaine instruction PRINT agira sur la ligne suivante.

— Une instruction PRINT avec une liste vide provoque un saut d'une ligne.

Exemple : Les séquences suivantes ont donc le même effet :



REMARQUE : Certains systèmes (PHILIPS-Hewlett Packard) permettent d'écrire :

PRINT A, B, C, /, X

ce qui provoquera l'édition d'une première ligne contenant les valeurs numériques de A, B, C puis le passage à la ligne suivante (rôle du SLASH) et enfin l'édition de la valeur numérique de X.

5.5. Les instructions PRINT USING et IMAGE.

L'instruction PRINT classique, même avec l'utilisation de la fonction TAB, ne permet pas une mise en page très souple. Cet inconvénient mineur pour des appli-

cations très simples, est devenu plus ennuyeux dès qu'il a fallu traiter des applications nécessitant l'édition de tableaux.

Pour combler cette lacune, les créateurs de BASIC se sont inspiré de FORTRAN en ajoutant deux instructions : PRINT USING et IMAGE, la première destinée à donner la liste de sortie et la seconde à décrire le découpage de la zone de sortie (on dira aussi l'image de la zone). Au chapitre 11 figurent des extensions apportées sur certains systèmes aux instructions PRINT USING et IMAGE

Cette instruction peut prendre selon les systèmes l'une des quatre formes données par le tableau suivant :

Formes	*Systèmes*
PRINT USING n, liste	Mark II, Philips, Hewlett Packard, etc.
PRINT USING ident., liste	Mark II, Hewlett Packard, etc.
PRINT USING chaîne, liste	Hewlett Packard, Data General, DEC, etc.
PRINT IN IMAGE n, liste	
PRINT IN IMAGE ident., liste	CDC, XDS, etc.

Dans ces formes :

n = constante entière indiquant le numéro de la ligne qui contient l'IMAGE.

ident. = identificateur de la variable caractère qui contient l'IMAGE.

chaîne = chaîne de caractères contenant l'IMAGE.

Le détail de syntaxe varie beaucoup d'un système à l'autre et seule la première forme est présentée au paragraphe 5.5.1 (complément en 11.10).

5.5.1. L'instruction PRINT USING.

Cette instruction diffère de l'instruction PRINT ,vue précédemment, par le fait qu'elle fait référence à une instruction IMAGE qui décrit la mise en page désirée.

La forme la plus courante de l'instruction PRINT USING est

PRINT USING *n* < liste de sortie >

dans laquelle :

n est une constante numérique indiquant le numéro de l'instruction IMAGE à laquelle on fait référence.

La liste de sortie est similaire à celles admises par l'instruction PRINT (excepté TAB). Le point virgule est souvent interdit (PHILIPS).

Quelques règles :

— UNE INSTRUCTION PRINT USING doit obligatoirement être associée à une instruction IMAGE. Mais plusieurs instructions PRINT USING peuvent être associées à une même instruction IMAGE.

— L'édition d'une liste d'une instruction PRINT USING commence à un début de ligne (sauf chez certains comme PHILIPS, si la liste se termine par une virgule).

5.5.2. L'instruction IMAGE.

Cette instruction doit commencer par le caractére *deux points* (:) ou pour Hewlett Packard par le « mot clé » IMAGE. Pour indiquer l'allure de la zone de sortie, on fera appel à des constantes caractères (ici de longueur quelconque) et à trois types de descripteurs de conversion destinés à indiquer le type de présentation à donner aux valeurs numériques. On retrouve les trois présentations possibles pour les nombres :

- entière,
- décimale,
- forme avec exposant.

Mais ici la forme ne sera pas choisie par l'ordinateur mais par l'utilisateur.

Exemple :

```
0080 PRINT USING 100,X,N
0100 :LA RACINE ##.### A ETE ØBTENUE APRES ## ITERATIØNS
```

X sera édité selon le descripteur # # · # # #
N sera édité selon le descripteur # #

— Le descripteur format entier est constitué d'un signe facultatif suivi d'un nombre quelconque de caractères #.

Exemple : # # #

— Le descripteur format décimal est constitué d'un signe facultatif suivi de un ou plusieurs caractères #, suivi(s) d'un point « décimal » et de un ou plusieurs caractères #.

Exemple : ## · ###

— Le descripteur format avec exposant est constitué d'un descripteur décimal suivi de quatre points d'exclamation.

Exemple : ## · ###!!!!

Les quatre points d'exclamation correspondent aux caractères suivants :

E ± *xx* → puissance de 10.

La signification de ces trois descripteurs est la suivante :

— le descripteur de format entier indique que la valeur numérique doit être éditée sous forme entière et le nombre de caractères utilisés indique la longueur de la zone à prendre pour cette édition,

— le descripteur de format décimal indique que la valeur numérique doit être éditée sous forme décimale et précise le nombre de caractères avant et après la virgule à utiliser,

— le descripteur de format avec exposant indique que la valeur numérique doit être éditée sous forme décimale avec exposant et précise le nombre de caractères avant et après la virgule à utiliser.

5.5.3. Règles relatives à l'instruction IMAGE.

Soit n le nombre de valeurs (numériques ou non) d'une liste de sortie et m le nombre de descripteurs de l'instruction IMAGE associée. Différents cas peuvent se présenter selon les valeurs relatives de n et m.

Si $n = m$, l'instruction IMAGE est exploitée une fois et une seule à chaque exécution de l'instruction PRINT USING qui y fait référence.

Si $n < m$, l'instruction IMAGE ne sera pas exploitée complètement : l'édition se fera conformément à l'image mais seulement jusqu'au premier descripteur inutilisé (les n premiers descripteurs sont utilisés et l'impression est stoppée juste avant le $n + 1^{eme}$).

Si $n > m$, une « image » seule ne permet pas de satisfaire la liste de sortie. Cette image sera exploitée autant de fois qu'il le faudra pour satisfaire à cette liste et à chaque fois l'impression reprend au début de la ligne suivante.

L'édition du signe – est faite chaque fois que la valeur numérique est négative. Le signe + est habituellement remplacé par un blanc lorsque la valeur est positive ou nulle. Cependant si le descripteur contient un signe +, ce signe est édité lorsque la valeur est positive.

REMARQUES :

1. Le système Time Sharing PHILIPS 9200 offre à l'utilisateur la possibilité de choisir le périphérique de sortie entre terminal ou imprimante rapide : pour cela il accepte outre la syntaxe habituelle des instructions PRINT et MAT PRINT une forme supplémentaire.

```
PRINT          [expression] liste d'E/S
PRINT USING    [expression] liste d'E/S
MAT PRINT      [expression] liste d'E/S
```

la valeur de l'expression indique le périphérique à utiliser (1 pour le télétype et 4 pour l'imprimante).

2. D'autres descripteurs sont disponibles sur certains systèmes (cf. 11.10).

5.6. *Exercices.*

5.6.1. Énoncés.

— *Edition de tableaux :* soit une fonction définie par

$$\text{DEF} \quad \text{FNA}(x) = \text{EXP}(-0.4 * X) * \text{COS}(3 * X)$$

écrire le programme permettant de tabuler cette fonction pour x variant de 0 à 10 par pas de 1, de telle sorte que l'édition ait l'allure suivante

X	Y
0	1
1	-.147690E 01
2	.213690E 01
3	-.302506E 01
4	.417964E 01
5	-.561338E 01
6	.727879E 01
7	-.900721E 01
8	.104062E 02
9	-.106918E 02
10	.842185E 01

X	Y
0	1.000000
1	-.663612
2	.431432
3	-.274427
4	.170371
5	-.102813
6	.059903
7	-.033307
8	.017290
9	-.007982
10	.002825

1er *cas :* en utilisant l'instruction PRINT et la fonction TAB

2e *cas :* en utilisant l'instruction PRINT USING.

— *Tracé de courbes :* Écrire un programme permettant de tracer par points la fonction précédente. L'utilisateur devra pouvoir introduire les paramètres suivants :

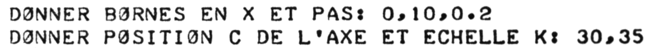
— bornes inférieure et supérieure de la variable x,

— pas de progression de x,

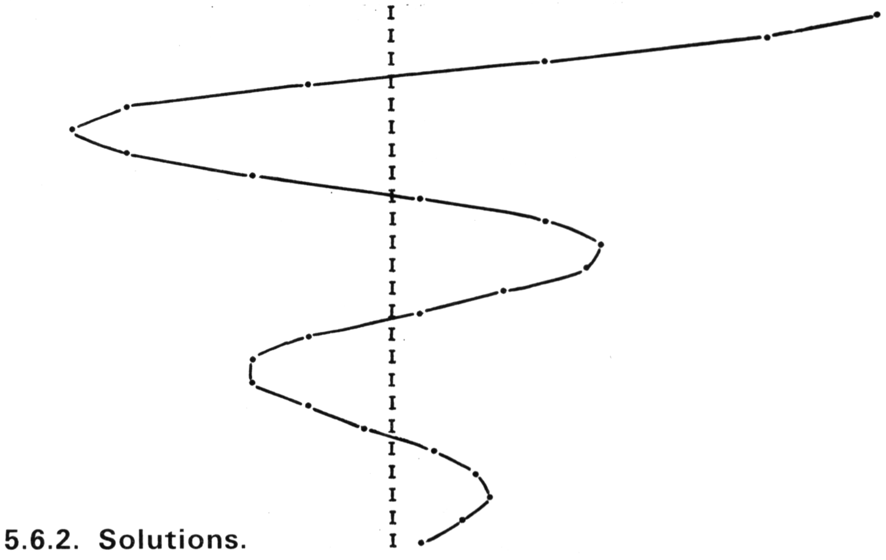
— position de l'axe des abscisses sur la feuille ; cet axe sera dans le sens vertical de défilement du papier,

— échelle des ordonnées.

Le résultat devra avoir l'allure suivante :



```
DØNNER BØRNES EN X ET PAS: 0,10,0.2
DØNNER PØSITIØN C DE L'AXE ET ECHELLE K: 30,35
```



5.6.2. Solutions.

— *Editions de tableaux :* Différentes méthodes permettent d'obtenir le résultat demandé. Les séquences suivantes ne constituent que des exemples.

```
0010 DEF FNA(X)=EXP(0.4*X)*CØS(3*X)
0020 GØSUB 200
0030 PRINT"I         X         I         Y         I"
0040 GØSUB 200
0050 FØR X=0 TØ 10
0055 Y=FNA(X)
0060 PRINT"I",X,"I",Y,"I"
0070 NEXT X
0080 GØSUB 200
0090 STØP
0200 PRINT "+-------------------+-------------------+"
0210 RETURN
0220 END
```

```
10 DEF FNA(X)=EXP(-0.4*X)*CØS(3*X)
20 GØSUB 100
30 PRINT USING 150
40 GØSUB 100
50 FØR X=0 TØ 10
60 PRINT USING 160,X,FNA(X)
70 NEXT X
80 GØSUB 100
90 STØP
100 PRINT"+--------+--------------+"
110 RETURN
150:I   X    I      Y      I
160:I  ###   I  ##.######   I
```

— *Tracé de courbes :* Rappelons que la fonction TAB a pour effet de tronquer l'argument (quand il n'est pas entier), puis d'affecter cette valeur à la position de la tête d'impression (72 caractères maximum par ligne).

Le tracé se fera à partir des calculs suivants :

— calcul de K . y, K étant l'échelle donnée par l'utilisateur,

— calcul de Ky + 0,5 puis calcul de z = entier (Ky + 0,5),

— la valeur de z ainsi obtenue constitue l'entier le plus proche de Ky et sert d'argument pour TAB(z).

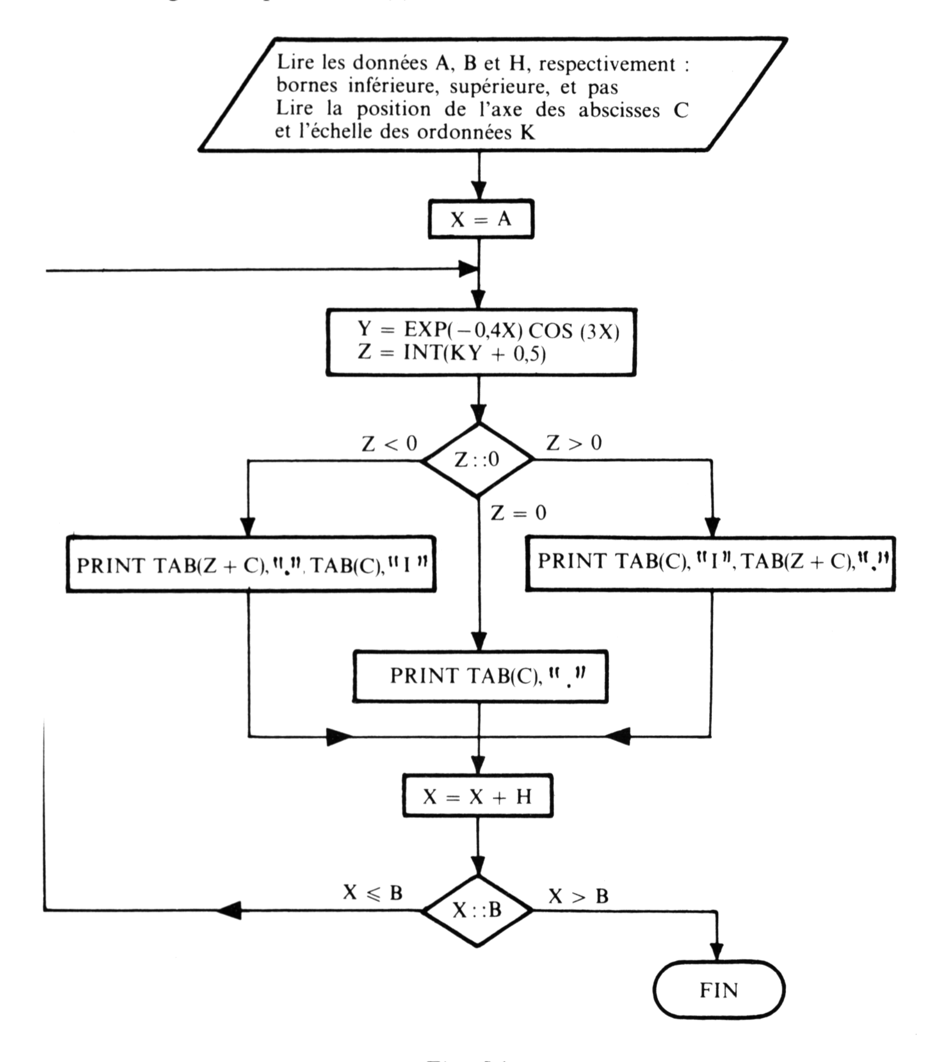


Fig. 5.1.

Le programme correspondant est donc le suivant :

```
0005 DEF FNA(X)=EXP(-0.4*X)*CØS(3*X)
0010 PRINT "DØNNER BØRNES EN X ET PAS:";
0015 INPUT A,B,H
0020 PRINT "DØNNER PØSITIØN C DE L'AXE ET ECHELLE K:";
0025 INPUT C,K
0030 FØR X=A TØ B STEP H
0035   LET Z=INT(K*FNA(X)+0.5)
0040   IF Z<0 THEN 60
0045   IF Z>0 THEN 70
0050   PRINT TAB(C),"."
0055   GØ TØ 75
0060   PRINT TAB(Z+C),".",TAB(C),"I"
0065   GØ TØ 75
0070   PRINT TAB(C),"I",TAB(Z+C),"."
0075 NEXT X
0080 END
```

CHAPITRE 6

FONCTIONS ET SOUS-PROGRAMMES

6.0. Introduction.

Nous avons vu au chapitre 1 (paragraphe 1.4.6) que la notion de sous-programme découlait, dans une certaine mesure, de la définition de fonction en mathématiques classiques. Plus précisément, pour éviter de faire répéter la description d'un procédé de calcul, les créateurs de BASIC ont adopté deux procédés élémentaires de définition de sous-programme :

— Définition de fonctions non standard.

— Notion de « subroutine ».

Une fonction, standard ou non, est destinée à donner, à partir d'une variable (parfois plusieurs variables), un résultat qui est une valeur numérique correspondant à la valeur de la fonction.

Une « subroutine » donne au contraire, pour résultat, une ou plusieurs valeurs numériques.

6.1. Les fonctions non standard.

L'utilisateur peut définir une fonction au moyen de l'instruction de déclaration DEF. Par exemple, l'instruction suivante :

DEF FNA (X) = (EXP (X) + EXP (– X))/2

permet de définir la fonction FNA (X) comme étant égale à :

$$\frac{e^x + e^{-x}}{2}.$$

Les noms des fonctions doivent tous commencer par le préfixe FN suivi d'une lettre de l'alphabet. L'utilisateur peut donc utiliser au plus

26 fonctions distinctes au sein d'un même programme. Cette limitation n'est pas gênante pour la majorité des applications redevables du BASIC.

L'utilisateur peut faire référence à cette fonction dans une expression arithmétique un peu comme en mathématiques. Par exemple, il pourra écrire :

$$\underbrace{U + FNA(Y + 3 * COS(Y))}_{\text{Expression arithmétique 1}} + \underbrace{FNA(Y + P2)}_{\text{Exp. 2}} - \underbrace{FNC(U)}_{\text{Exp. 3}} + \ldots$$

Ceci signifie qu'il faut évaluer la valeur mathématique des paramètres effectifs de FNA (ici Y + 3 COS Y) puis utiliser cette valeur pour « évaluer » la valeur numérique de la fonction FNA. Ensuite, il faudra répéter cette opération pour FNA (Y + P2) puis pour l'autre fonction FNC (U). Ensuite, on peut calculer la valeur numérique de l'expression arithmétique.

REMARQUE : Dans la pratique, on effectue les opérations dans l'ordre indiqué au paragraphe 1.4.9.1 ce qui ne change rien au résultat final.

6.1.1. Règles relatives aux fonctions non standard.

— Toute fonction non standard doit pouvoir être définie au moyen d'une expression arithmétique (voir extension au chapitre 11).

— Ces fonctions ne peuvent dépendre que d'un seul paramètre formel.

REMARQUE : Certains compilateurs, par exemple le BASIC des ordinateurs XDS SIGMA 5/7 (commercialisés en France sous le nom de CII 10070), DEC Pdp 11 XDS 940, CDC, etc. acceptent des fonctions ayant plusieurs paramètres (cf. paragraphe 11.4).

— Une fonction doit être définie une seule fois dans un programme.

— Il n'est pas nécessaire qu'elle soit définie avant qu'on y fasse référence.

— Pour définir une fonction non standard, on peut faire appel à une autre fonction non standard à condition que ceci n'entraîne pas une « récursivité ».

L'exemple qui suit satisfait à ces règles :

```
0010 DEF FNA(X)=(EXP(X)+EXP(-X))/2
0020 DEF FNB(X)=SQR(FNA(X)↑2-1)
```

En effet :

- la définition de FNA fait appel à une fonction standard et
- FNB fait appel à FNA.

Les deux exemples qui suivent correspondent à des fautes de programmation (1) :

Premier exemple :

```
0010 DEF FNA(X)=X*FNA(X-1)
```

La définition de FNA fait appel à FNA : on dit qu'il y a « récursivité d'appel ».

Deuxième exemple :

```
0050 DEF FNA(X)=X+FNB(X+...)+...
0060 DEF FNB(Y)=Y+FNC(X+...)+...
0070 DEF FNC(Z)=Z*FNA(.....)-...
```

Les définitions des fonctions s'opèrent comme suit :

FNA fait appel à FNB
FNB fait appel à FNC
FNC fait appel à FNA.

On dit qu'il y a une « référence circulaire » (ou « récursivité croisée »).

Remarques :

1. La récursivité pour les fonctions est interdite sur la plupart des systèmes car elle complique la réalisation d'un compilateur, ce qui est contraire à l'objectif du BASIC. Par contre, elle est souvent autorisée pour les subroutines.

2. Souvent les compilateurs ne vérifient pas s'il y a référence croisée et les erreurs apparaissent à l'exécution des programmes.

6.1.2. Exemple d'utilisation :

Tabuler la fonction $\frac{e^x + e^{-x}}{2}$ pour x variant de 0 à 1 par pas de 0,1 et pour x variant de 10 à 15 par pas de 0,2.

Première solution

```
0010 FØR X=0 TØ 1 STEP 0.1
0020    PRINT X;(EXP(X)+EXP(-X))/2
0030 NEXT X
0040 FØR X=10 TØ 15 STEP 0.2
0050    PRINT X;(EXP(X)+EXP(-X))/2
0060 NEXT X
0070 END
```

Deuxième solution

```
0010 DEF FNA(X) = (EXP(X)+EXP(-X))/2
0020 FØR X=0 TØ 1 STEP 0.1
0030    PRINT X;FNA(X)
0040 NEXT X
0050 FØR X=10 TØ 15 STEP 0.2
0060    PRINT X;FNA(X)
0070 NEXT X
0080 END
```

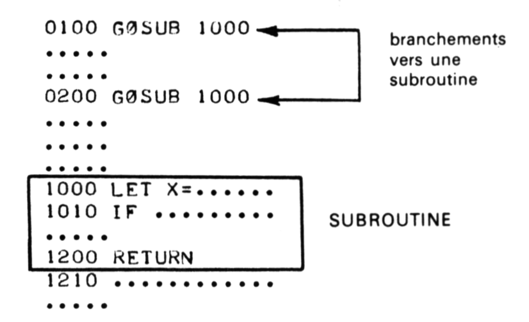
(1) Sauf sur certains systèmes (voir à ce sujet le paragraphe 11.4).

6.2. Les sous-programmes « subroutines ».

La forme initiale de ces sous-programmes très simple s'est avérée peu à peu insuffisante et des extensions variées ont été apportées. Dans ce livre figurent donc la forme initiale et les extensions les plus intéressantes.

6.2.1. Forme élémentaire des subroutines.

La forme la plus simple consiste à écrire dans le corps même du programme une paire d'instructions qui constitueront une subroutine. La forme est donc la suivante :

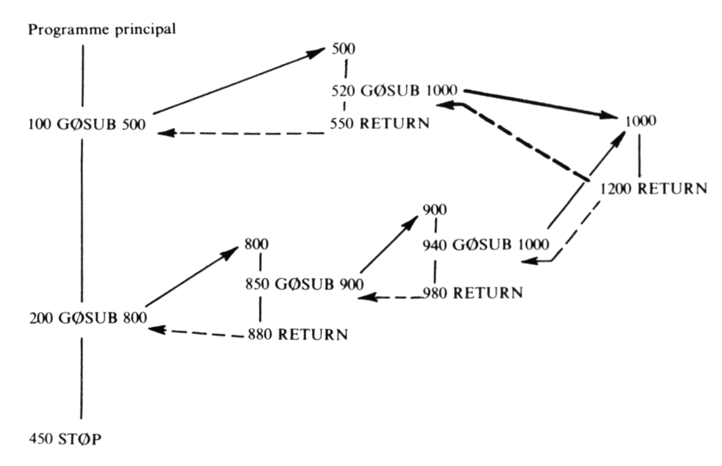


Une subroutine est donc une portion de programme se terminant par au moins une instruction RETURN.

Le branchement vers une subroutine se fait au moyen de l'instruction GOSUB suivi du numéro de ligne de la prochaine instruction à exécuter.

L'instruction RETURN provoque le retour à l'instruction qui suit immédiatement l'instruction GOSUB ayant provoqué le branchement.

6.2.2. Quelques règles relatives aux subroutines.



1) Avant de rencontrer une instruction RETURN, il faut au préalable avoir exécuté au moins une instruction GOSUB.

2) Une subroutine peut elle-même appeler une autre subroutine et ainsi de suite. L'exemple suivant montre comment cette possibilité s'exploite.

Cet exemple montre que l'enchevêtrement des soubroutines peut être assez compliqué.

Il est vivement déconseillé de sortir d'une subroutine autrement que par une instruction RETURN.

3) Sur la majorité des systèmes (CIGI-TYMSHARE, Pdp, CDC, VARIAN) la récursivité est autorisée. Ceci signifie qu'une subroutine peut s'appeler elle-même, que deux subroutines peuvent s'appeler mutuellement (on dit alors récursivité croisée), etc.

L'intérêt de la récursivité n'apparaît que pour des applications peu fréquentes dont le problème de la « Tour de Hanoi » constitue un excellent exemple.

4) Une subroutine peut avoir plusieurs points d'entrée à condition toutefois, que ceci n'entraîne pas une violation de la règle 1. Cette possibilité qui n'ajoute rien à la « puissance du langage » permet certaines astuces de programmation dont l'exemple suivant ne constitue qu'une illustration.

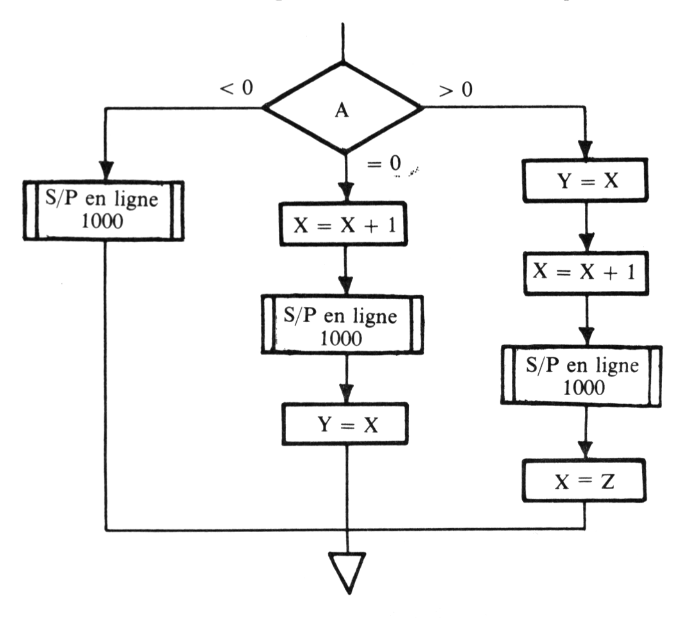
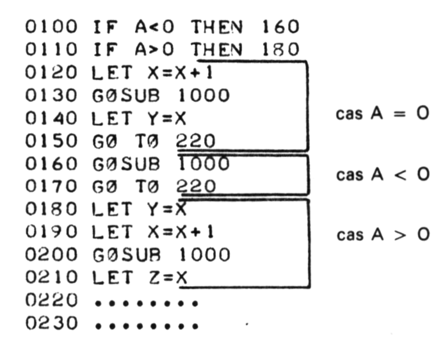


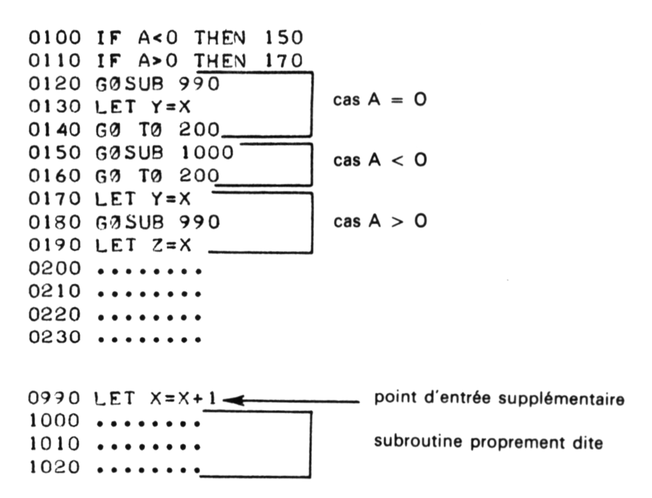
FIG. 6.1.

Avec ce qui précède cette portion d'organigramme se traduit par la séquence suivante :



```
0100 IF A<0 THEN 160
0110 IF A>0 THEN 180
0120 LET X=X+1
0130 GØSUB 1000
0140 LET Y=X
0150 GØ TØ 220
0160 GØSUB 1000
0170 GØ TØ 220
0180 LET Y=X
0190 LET X=X+1
0200 GØSUB 1000
0210 LET Z=X
0220 ........
0230 ........
```


En utilisant la possibilité signalée en 4, nous pourrons écrire :



```
0100 IF A<0 THEN 150
0110 IF A>0 THEN 170
0120 GØSUB 990
0130 LET Y=X
0140 GØ TØ 200
0150 GØSUB 1000
0160 GØ TØ 200
0170 LET Y=X
0180 GØSUB 990
0190 LET Z=X
0200 ........
0210 ........
0220 ........
0230 ........

0990 LET X=X+1
1000 ........
1010 ........
1020 ........
```


6.2.3. Instruction GOSUB calculée :

De nombreux systèmes ne disposent pas de cette instruction. Elle est disponible notamment sur les ordinateurs VARIAN 620, Pdp 11, CDC. XDS 940.

Sa forme varie évidemment d'un constructeur à l'autre (comme l'instruction GO TO calculée) :

Forme VARIAN GØSUB E ØF $s_1, s_2, ..., s_n$

Forme — Pdp 11 ØN E GØSUB $s_1, s_2, ..., s_n$

— CDC

— XDS 940

dans laquelle E représente une expression et $s_1, s_2, ..., s_n$ des numéros de ligne.

Selon la valeur de la partie entière de E, il y aura branchement à l'instruction de la ligne s_1, ou s_2 ..., etc. Le retour se fera ensuite par l'intermédiaire d'une instruction RETURN à la ligne suivante.

Exemple d'utilisation : Pour une compagnie d'assurances un conducteur automobile est caractérisé par trois variables : B1, B2, B3 susceptibles de prendre chacune deux valeurs ayant la signification suivante :

B1 = 0 conducteur âgé de moins de 25 ans,
 1 conducteur âgé de plus de 25 ans
B2 = 0 conducteur célibataire
 1 conducteur marié
B3 = 0 n'a pas eu d'accident depuis au moins un an
 1 a eu au moins un accident depuis un an.

Selon la valeur de chaque variable il faut se brancher vers le sous-programme de calcul de la prime correspondante.

Le tableau suivant indique les numéros de ligne des sous-programmes auxquels il faudra se brancher :

*B*1	*B*2	*B*3	*Numéro de ligne*
0	0	0	400
0	0	1	450
0	1	0	480
0	1	1	500
1	0	0	520
1	0	1	800
1	1	0	600
1	1	1	600

Écrire la séquence d'instructions permettant d'effectuer le branchement vers le bon sous-programme.

Réponse : Si nous comptons en binaire, en affectant respectivement à B1, B2 et B3 les poids 2^2, 2^1, 2^0, le triplet B1, B2, B3 prend l'une des valeurs suivantes : 0, 1, 2, 3, 4, 5, 6 ou 7.

Par conséquent, nous pouvons construire une expression dont les valeurs correspondantes iront de 1 à 8. Cette expression est :

$$B3 + 2B2 + 4B1 + 1.$$

Nous pouvons donc écrire :

```
GØSUB B3 + 2 * B2 + 4 * B1 + 1 ØF 400, 450, 480, 500, 520, 800, 600, 600
```

ou

```
ØN B3 + 2 * B2 + 4 * B1 + 1 GØSUB 400, 450, 480, 500, 520, 800, 600, 600
```

6.3. Exercices.

1) Écrire le programme correspondant à la recherche des nombres premiers selon la méthode exposée en 1.6.2 et 1.6.3. On se limitera à la recherche des cent premiers nombres premiers.

2) Écrire deux subroutines permettant de calculer factorielle n :

Première subroutine calcul normal : $n! = 2 \times 3 \times 4 \times ... \times n$.

Deuxième subroutine calcul à partir de la relation de récurrence $n! = n(n-1)!$

Cette subroutine fera donc appel à la récursivité.

Pour chaque subroutine, on fera l'hypothèse que N est entier, positif ou nul. La valeur de la factorielle sera rangée dans une variable F.

1. *Solutions*

```
0010 DIM T(3000)
0020 LET T(1)=1
0030 LET T(2)=2
0040 LET T(3)=3
0050 LET T(4)=5
0070 LET I=3
0080 INPUT N
0100 FØR L=1 TØ N
0110 LET A=6*L-1
0120 GØSUB 200
0130 LET A=A+2
0140 GØSUB 200
0150 NEXT L
0160 PRINT "LA LISTE SUIVANTE CØNTIENT";I;"NØMBRES PREMIERS"
0161 PRINT
0162 FØR J=1 TØ I
0165 PRINT T(J);
0170 NEXT J
0175 STØP
0200 FØR J=4 TØ I
0210 LET U=T(J)
0220 IF U*U>A THEN 260
0230 LET R=A-INT(A/U)*U
0240 IF R=0 THEN 280
0250 NEXT J
0260 LET I=I+1
0270 LET T(I)=A
0275 IF I=3000 THEN 160  ) test pour éviter de déborder du tableau.
0280 RETURN
0290 END
```

Valeur de N frappée à l'exécution

```
30
LA LISTE SUIVANTE CØNTIENT     43 NØMBRES PREMIERS

   1     2     3     5     7    11    13    17    19    23    29    31
  37    41    43    47    53    59    61    67    71    73    79    83
  89    97   101   103   107   109   113   127   131   137   139   149
 151   157   163   167   173   179   181
STØP AT LINE  0175
```

2. *Première subroutine :*

```
090 LET N=.....
100 GØSUB 200

     .
     .
     .
200 LET F=1
210 IF N<=1 THEN 250
220 FØR I=2 TØ N
230 LET F=F*I
240 NEXT I
250 RETURN
```

Deuxième subroutine :

```
100 GØSUB 200
     .
     .
     .
     .
200 LET F=1
210 IF N<=1 THEN 250
220 LET F=F*N
230 LET N=N-1
240 GØSUB 210
250 RETURN
```

REMARQUE : Avec certains systèmes, tels que nous le verrons lors du chapitre 11 nous pourrons écrire :

```
200 LET F=1
210 IF N<=1 THEN RETURN
220 LET F=F*N
230 LET N=N-1
240 GØSUB 210
```

CHAPITRE 7

LES INSTRUCTIONS DE TRAITEMENT DES TABLEAUX

7.0. Introduction : une particularité intéressante du basic.

On désigne fréquemment un ensemble de quantités de même nature par un nom (identificateur) unique, auquel on ajoute des indices. Tel est le cas, en mathématiques par exemple, pour :

- Les n composantes d'un vecteur V dans un espace à n dimensions qui seront dénommées : $v(1)$, $v(2)$ ... $v(n)$.
- Les $n \times p$ éléments d'une matrice M qui seront dénommés :

$$m(1, 1),\ m(1, 2), ..., m(1, p),\ m(2, 1),\ m(2, 2) ...\ m(n, p).$$

A l'aide des instructions des paragraphes précédents, l'utilisateur peut réaliser des boucles de programmes pour effectuer sur ces variables, dans un ordre choisi par lui, des opérations d'Entrée-Sortie et des calculs.

Il peut aussi employer les instructions synthétiques du BASIC pour effectuer très simplement, sur des matrices et des tableaux, les opérations d'Entrée-Sortie et de calcul les plus courantes.

7.0.1. Terminologie.

Dans le cours du chapitre, nous utiliserons un ensemble de termes dont la signification est rappelée ici sous une forme qui est destinée à faciliter la lecture de l'exposé.

Élément de tableau : variable définie par un identificateur, suivi des indices entre parenthèses ;
les instructions de traitement des tableaux s'appliquent pour un nombre d'indices limité à deux.

Tableau : contient l'ensemble des éléments précédents ;
il se définit par un identificateur.

Matrice : tableau numérique à deux indices, sur lequel s'appliquent des opérations répondant aux règles de l'algèbre linéaire.

Variable indicée : variable dont l'identificateur peut être suivi d'indices (1) ; ce terme, peu précis, désigne soit un élément de tableau, soit un tableau.

Vecteur : matrice particulière ayant une seule ligne ou une seule colonne.

(1) Le chapitre 11 apporte des précisions sur le nombre et la plage de variation des indices.

7.0.2. Forme générale des instructions.

Les instructions sur tableaux ou sur matrices — l'instruction DIM exceptée — doivent toujours être précédées du préfixe MAT (abréviation de MATRIX).

Les identificateurs ne comportent qu'une seule lettre (cf. chap. 2).

7.0.3. Classification des instructions.

Les instructions de traitement des tableaux peuvent se classer en trois catégories.

Catégorie	*Fonction*	*Exemple*	*Remarques*
Instructions d'entrée et de sortie	Entrée statique	MAT READ A(I, J)	(2)
	Entrée dynamique	MAT INPUT A(I, J)	(1) et (2)
	Entrée à partir d'un fichier	MAT INPUT 1, A(28, 34)	(1) et (4)
	Sortie	MAT PRINT A	
	Sortie en format condensé	MAT PRINT A ;	
	Sortie avec format	MAT PRINT USING 120, A	(1)
	Sortie vers un fichier	MAT PRINT 1, A(28, 34)	(1) et (4)
Instructions de calcul matriciel	Addition	MAT A = B + C	(3)
	Soustraction	MAT A = B − C	(3)
	Multiplication par un scalaire	MAT A = (K) * B	(3)
	Produit	MAT A = B * C	(3)
	Transposition	MAT A = TRN(B)	(3)
	Inversion	MAT A = INV(B)	(3)
Instructions d'initialisation	Équivalence	MAT A = B	
	Constante	MAT A = CON(I, J)	(2)
	Mise à zéro	MAT A = ZER(I, J)	(2)
	Matrice unité	MAT A = IDN(I, I)	(2)

FIG. 7.1.

REMARQUES

(1) Ces instructions ne sont pas offertes par tous les compilateurs.

La sortie avec format est traitée au chapitre des extensions (cf. 11.10.3).

(2) Ces instructions permettent de dimensionner les tableaux, si l'utilisateur le désire. Ce point est examiné au paragraphe 7.5.

(3) Les instructions de calcul matriciel sont des instructions d'affectation (cf. 1.4.3) qui opèrent en suivant les règles de l'algèbre linéaire. Mais dans certains cas (inversion de matrice par exemple), le compilateur ne range pas à la fois la matrice « donnée » (dont le nom figure à droite du signe « égal ») et la matrice résultat (dont le nom figure à gauche du signe « égal »), dans les mêmes emplacements de la mémoire.

(4) Ces instructions sont traitées au chapitre des « Fichiers » (cf. 10.3.2.3).

Il en résulte des règles différentes pour les diverses instructions (ces règles varient par ailleurs d'un constructeur à l'autre), qui sont réunies dans le tableau ci-dessous.

Nature de l'instruction	*Possibilité de faire figurer le même nom de matrice des deux cotés du signe « égal »*
Addition et soustraction	oui
Multiplication par un scalaire.	souvent
Produit	non
Transposition	non
Inversion	quelquefois

FIG. 7.2.

7.1. Instructions DIM et option BASE

L'instruction de déclaration DIM permet de déclarer une liste de variables indicées dont le type peut être numérique ou caractère (cf. paragraphes 2.3.1, 2.3.2 et 2.3.3). La forme syntaxique de cette instruction est habituellement :

⟨ numéro de ligne ⟩ DIM ⟨ liste ⟩

La liste contient une suite d'identificateurs de tableaux avec leurs dimensions, séparés par une virgule.

Chaque identificateur est suivi entre parenthèses, des valeurs maximum que peuvent prendre les indices. Pour les tableaux à deux dimensions, le premier indice est l'indice ligne (ou numéro de ligne), le second est l'indice colonne.

Exemple : 20 DIM A(3,4), B(6)

Sans autre précision l'instruction DIM A(3,4), B(6) signifie que A est un tableau dont le premier indice (indice ligne) varie de 0 à 3 et le second indice de 0 à 4. De même B est un tableau à une dimension dont l'indice varie de 0 à 6.

L'instruction OPTION BASE *n* où *n* vaut 0 ou 1 indique si les indices doivent partir de 0 ou de 1. Cette instruction qui doit précéder les instructions DIM est d'origine récente et n'est pas supportée par tous les systèmes. Pour les systèmes qui l'acceptent, cette instruction est optionnelle ; en son absence les indices partent de 0.

Cependant dans tous les cas les instructions matricielles ne portent pas sur les indices 0 lorsqu'ils existent. Ceci permet d'améliorer la compatibilité des deux interprétations.

7.2. Instructions d'entrée et de sortie pour les tableaux.

On peut réaliser l'entrée ou la sortie des éléments des tableaux en utilisant les instructions :

READ A(I, J) ou INPUT A(I, J)
PRINT A(I, J)

De cette manière, l'utilisateur a toute latitude sur la présentation des données, en faisant progresser les indices dans une boucle de calcul qui est spécifique de son programme.

Il peut aussi accepter les règles de progression des indices propres aux instructions d'Entrée/Sortie des tableaux et éviter de programmer lui-même la boucle de calcul.

7.2.1. L'instruction d'entrée.

L'instruction d'entrée est de la forme :

MAT READ ⟨ liste ⟩ ou MAT INPUT ⟨ liste ⟩ (1)

La liste contient :

- une suite d'identificateurs de tableaux sans indices, ou
- une suite d'identificateurs de tableaux avec indices, ou
- une suite d'identificateurs de tableaux avec et sans indices.

On écrit, par exemple :

MAT READ A(2,3), B(4) ou MAT INPUT A(2,3), B(4).

Dans ces instructions :

- A et B sont les noms de deux tableaux,
- le tableau A a deux lignes et trois colonnes,
- le tableau B reçoit ici un seul indice qui peut prendre quatre valeurs (1, 2, 3 et 4).

(1) La seule différence entre ces deux formes d'entrée est que la première porte sur des données statiques, la seconde sur des données dynamiques (cf. 5.2).

Au moment de la lecture, les données sont attribuées aux éléments :
pour une variable indicée : dans l'ordre croissant de l'indice,
pour un tableau : ligne par ligne (l'indice colonne varie avant l'indice ligne).
Les données doivent être des constantes numériques.

7.2.2. L'instruction de sortie.

L'instruction de sortie est de la forme :

MAT PRINT A, B

Dans cette instruction, A et B sont les noms des tableaux. On remarquera qu'il n'y a pas lieu de préciser les dimensions des tableaux dans cette instruction.

On peut également écrire :

MAT PRINT A; B pour que A soit imprimé en format condensé,
ou MAT PRINT A, B; pour que B soit imprimé en format condensé,
ou MAT PRINT A; B; pour que A et B soient imprimés en format condensé.

7.2.3. Exemples d'emploi des instructions d'entrée et de sortie.

7.2.3.1. Instruction d'entrée sans dimension.

Dans cet exemple, la déclaration DIM affecte les dimensions à A et à B. On remarque par ailleurs que le tableau B est présenté en format condensé.

```
0010 DIM A(2,3),B(4)
0020 MAT READ A,B
0030 PRINT "TABLEAU A"
0040 MAT PRINT A
0050 PRINT "LISTE B"
0060 MAT PRINT B;
0070 END
0080 DATA 1,2,3,4,5,6,40,41,42,43
```

```
TABLEAU A
             1                  2                  3
             4                  5                  6
LISTE B
    40    41    42    43
```

Résultat de l'exécution

7.2.3.2. Instruction d'entrée avec dimensions.

Ici, le compilateur réserve les emplacements de mémoire nécessaires pour un tableau de 20 lignes et 20 colonnes.

Mais les dimensions effectivement prises en compte sont définies dans l'instruction READ.

Ces dernières doivent être au plus égales à celles de la déclaration DIM (1).

```
0010 DIM A(20,20)
0020 MAT READ A(2,3)
0030 MAT PRINT A;
0040 END
0050 DATA 1,2,3,4,5,6

     1      2      3        } Résultat de
     4      5      6        } l'exécution
```

7.3. Les instructions de calcul matriciel.

Dans le paragraphe précédent, nous avons utilisé les instructions d'entrée et de et de sortie pour des tableaux, sans nous soucier de leur nature mathématique (pour cette raison, nous avons préféré la dénomination « tableaux »).

Les instructions qui vont suivre, par contre, s'appliquent uniquement à des matrices et à des vecteurs. Chaque instruction permet d'effectuer une opération et une seule (addition, multiplication, division, etc.). Une opération complexe telle que

$$A = BC - D^{-1}E$$

devra donc être réalisée à l'aide de plusieurs instructions successives.

Pour chaque instruction décrite les règles mathématiques correspondantes du calcul matriciel sont rappelées.

Un élément d'une matrice A est désigné par a_{ij} où i est le numéro de la ligne et j le numéro de la colonne.

7.3.1. Addition et soustraction de deux matrices.

Ces instructions s'écrivent :

- pour l'addition : MAT A = B + C
- pour la soustraction : MAT A = B − C

Ces opérations n'ont de sens que si les trois matrices A, B et C ont les mêmes nombres de lignes et de colonnes.

(1) Le paragraphe 7.5 apporte des précisions à ce sujet.

Premier exemple : addition de deux matrices.

```
0010 DIM A(2,3),B(2,3),C(2,3)
0020 MAT READ A,B,C
0030 MAT A=B+C
0040 MAT PRINT A
0050 STØP
0060 DATA 0,0,0,0,0,0,1,2,3,4,5,6,0.1,0.2,0.3,0.4,0.5,0.6

.110000E 01    .220000E 01    .330000E 01 } Résultat de
.440000E 01    .550000E 01    .660000E 01 } l'exécution
```

Deuxième exemple : soustraction de deux matrices ; le résultat est rangé à la place d'une des deux matrices « données ».

```
0010 DIM A(2,2),B(2,2)
0020 MAT READ A,B
0030 MAT A=A-B
0040 MAT PRINT A;
0050 STØP
0060 DATA -1,2,4,7,-9,6,-4,8

   8  -   4  } Résultat de
   8  -   1  } l'exécution
```

7.3.2. Multiplication d'une matrice par un scalaire.

L'instruction s'écrit : MAT A = (K) * B.

Dans cette instruction :

- A et B sont les noms des deux matrices,
- K est un scalaire (constante, variable ou expression arithmétique) et doit être mis entre parenthèses pour faire la distinction avec l'éventualité où K représenterait une matrice.

Cette instruction s'exécute de la façon suivante :

- l'expression (quand K est une expression) est d'abord calculée et donne un résultat k,
- tous les éléments de B sont ensuite multipliés par k.

Le résultat est donc : $A = kB$ soit encore

$$a_{ij} = kb_{ij}.$$

La matrice A, résultat de l'opération kB a les mêmes dimensions que B. Les matrices A et B doivent donc être définies comme de mêmes dimensions.

Exemple :

```
0010 DIM A(2,3),B(2,3)
0020 READ X,Y,Z
0030 MAT READ B
0040 MAT A=((X↑2+Y)/Z)*B
0050 PRINT "    MATRICE B"
0060 MAT PRINT B;
0070 PRINT "    MATRICE PRØDUIT A"
0080 MAT PRINT A
0090 STØP
0100 DATA 2,3,2
0110 DATA 1,2,3,4,5,6
```

```
     MATRICE B
     1       2       3
     4       5       6
                                                     Résultat de
     MATRICE PRØDUIT A                               l'exécution
 .350000E 01              7      .105000E 02
          14     .175000E 02              21
```

7.3.3. Produit de deux matrices.

L'instruction s'écrit : MAT A = B * C.

Dans cette instruction, A, B et C sont les noms des matrices. Le résultat est : A = B . C ou sous forme développée :

$$a_{ij} = \sum_{l=1}^{l=n} b_{il} c_{lj}.$$

Exemple :

```
0010 DIM A(1,3),B(3,3),C(1,3)
0020 MAT READ A,B
0030 MAT C=A*B
0040 PRINT "    MATRICE A"
0050 MAT PRINT A;
0060 PRINT "    MATRICE B"
0070 MAT PRINT B;
0080 PRINT "MATRICE PRØDUIT : C"
0090 MAT PRINT C;
0100 STØP
0110 DATA 1,2,-3,4,8,5,0,-1,1,1,0,-4
```

```
     MATRICE A
     1       2 -     3

     MATRICE B                  Résultat de
     4       8       5          l'exécution
     0 -     1       1
     1       0 -     4

 MATRICE PRØDUIT : C
     1       6      19
```

Le produit de deux matrices s'applique à des matrices rectangles. Mais dans ce cas le produit est défini seulement si le nombre de colonnes de la première matrice est égal au nombre de lignes de la seconde.

On doit avoir : $A_{mp} = B_{mn}\ C_{np}$.

L'utilisateur doit donc affecter les dimensions de B, C et aussi de la matrice produit A avec attention.

7.3.4. Transposition d'une matrice.

L'instruction s'écrit :

MAT A = TRN(B).

Dans cette instruction, A et B sont les noms des matrices.

Le résultat est : $A = B^T$ ou encore
$a_{ij} = b_{ji}$.

La matrice transposée s'obtient en échangeant lignes et colonnes de la matrice origine.

Les dimensions des matrices doivent naturellement se correspondre ; on doit avoir en effet : DIM A(M, N), B(N, M).

Exemple :

```
0010 DIM A(2,3),B(3,2)
0020 MAT READ B
0030 PRINT "    MATRICE B"
0040 MAT PRINT B;
0050 MAT A=TRN(B)
0060 PRINT "    MATRICE TRANSPØSEE A"
0070 MAT PRINT A;
0080 STØP
0090 DATA 1,2,3,4,5,6
```

```
MATRICE B
1        2
3        4
5        6

MATRICE TRANSPØSEE A
1        3        5
2        4        6
```

Résultat de l'exécution

7.3.5. Inversion de matrice.

L'instruction s'écrit :

MAT A = INV(B).

Dans cette instruction, A et B sont les noms des matrices.

REMARQUE 1 : Conditions de validité de l'opération. On sait que les éléments de A sont donnés par la formule $a_{ij} = \frac{B_{ij}}{\Delta}$, dans laquelle Δ est le déterminant de B et B_{ij} le mineur relatif à l'élément b_{ij}. Du strict point de vue mathématique il en résulte que si la matrice est dégénérée (matrice dont le déterminant est nul), l'opération n'est pas définie.

L'analyse numérique nous enseigne en plus que si Δ est petit par rapport aux coefficients de B (on dit que B est « mal conditionnée »), le calcul en ordinateur entraîne des erreurs d'arrondis qui, cumulées, risquent de rendre le résultat non significatif.

L'emploi de cette instruction implique donc que le programmeur s'est assuré au préalable de la validité de l'opération.

Exemple :

```
0010 DIM A(3,3)
0020 MAT READ A
0030 PRINT "    MATRICE A"
0040 MAT PRINT A;
0050 MAT A=INV(A)
0060 PRINT "    MATRICE INVERSE"
0070 MAT PRINT A
0080 STØP
0090 DATA 1,2,3,4,5,6,7,8,9
```

```
     MATRICE A
     1        2        3
     4        5        6
     7        8        9

     MATRICE INVERSE
-.536871E 09       1073741823      -  536870913
  1073741811     -.214748E 10      .107374E 10
-.536871E 09      .107374E 10      -  536870915
```

Résultat de l'exécution

REMARQUE 2 : Estimation des erreurs d'arrondi. Sans développer les méthodes mathématiques portant sur ce sujet, montrons une façon simple permettant d'estimer l'importance des erreurs d'arrondi.

Soit à inverser une matrice A.

Le résultat donné par le calculateur est $B \# A^{-1}$.

Formons $P = AB$, puis calculons

$E = AB - U$, U étant la matrice unité.

On peut remarquer que E(1, 2) et E(1, 3) sont de l'ordre de 10^{-7}

alors que B(3, 2) est de l'ordre de 10^{-6}.

L'erreur sur AB est 10 % de l'élément le plus petit de B.

Une analyse plus serrée du résultat est donc nécessaire.

Exemple :

```
0010 DIM A(3,3),B(3,3),U(3,3),P(3,3),E(3,3)
0020 MAT U=IDN
0030 MAT READ A
0040 PRINT "  MATRICE A"
0050 MAT PRINT A
0060 MAT B=INV(A)
0070 PRINT "  MATRICE B = INVERSE DE A"
0080 MAT PRINT B
0090 MAT P=A*B
0100 PRINT "  ESTIMATIØN DES ERREURS D'ARRØNDI"
0110 MAT E=P-U
0120 MAT PRINT E
0130 STØP
0140 DATA 0.8,0.6,0.36,0.4,0.28,0.2,0.4,0.3,0.25
```

```
  MATRICE A
 .800000E 00     .600000E 00     .360000E 00
 .400000E 00     .280000E 00     .200000E 00
 .400000E 00     .300000E 00     .250000E 00

  MATRICE B = INVERSE DE A
-.892857E 01     .375000E 02    -.171429E 02
 .178571E 02    -.500000E 02     .142857E 02
-.714286E 01     .332615E-06     .142857E 02

  ESTIMATIØN DES ERREURS D'ARRØNDI
 .372529E-08    -.292701E-07     .298023E-07
           0     .558794E-08     .372529E-08
-.372529E-08     .864799E-08     .372529E-08
```

Résultat de l'exécution

REMARQUE 3 : Pour faciliter le diagnostic, certains compilateurs (DEC par exemple) fournissent la valeur du déterminant de la matrice inversée, une fois l'inversion réalisée.

Ainsi le compilateur BASIC du Pdp 11 permet d'écrire :

```
100 MAT A = INV(X) : D1 = DET
110 MAT B = INV(Y) : D2 = DET
120 IF D1 > D2 GO TO 200
```

D1 et D2 sont respectivement les déterminants de A et de B.

7.4. *Les instructions d'initialisation pour les tableaux.*

Le paragraphe 3.2 a expliqué les différentes méthodes d'initialisation des variables.

On a vu en particulier que l'initialisation pouvait être (ou ne pas être) faite par le compilateur. Il en est de même pour les éléments des tableaux.

L'initialisation par lecture de données est particulièrement fastidieuse quand les éléments d'un grand tableau doivent être, initialement, tous égaux.

Les instructions synthétiques du langage BASIC permettent de fixer aisément leur valeur initiale, ou de la modifier dans le cours d'un programme.

7.4.1. Équivalence de deux tableaux.

L'instruction s'écrit :

MAT A = B.

Dans cette instruction, A et B sont les noms des tableaux.

Le résultat est $a_{ij} = b_{ij}$.

Les deux tableaux doivent être de dimensions identiques.

Exemple :

```
0010 DIM A(2,3),B(2,3)
0020 MAT READ B
0030 MAT A=B
0040 MAT PRINT A;
0050 STØP
0060 DATA 1,2,3,4,5,6

     1     2     3   } Résultat de
     4     5     6   } l'exécution
```

7.4.2. Instruction « constante ».

L'instruction s'écrit : 05 MAT A = CØN ou
10 MAT B = CØN(I) ou
15 MAT C = CØN(I, J)

Dans tous les cas, tous les éléments de tableau reçoivent la valeur 1.

L'instruction permet en plus de préciser les dimensions du tableau (lignes 10 et 15 de l'exemple).

Exemple :

```
0010 DIM A(20,20)
0020 MAT A=CØN(2,3)
0030 MAT PRINT A;
0040 STØP

     1     1     1   } Résultat de
     1     1     1   } l'exécution
```

7.4.3. Instruction de remise à zéro.

L'instruction s'écrit : 05 MAT A = ZER ou
10 MAT B = ZER(I) ou
15 MAT C = ZER(I, J).

Dans tous les cas, tous les éléments du tableau reçoivent la valeur zéro.

Les règles sur les dimensions sont les mêmes que pour l'instruction constante.

Exemple :

```
0010 DIM A(20,20)
0020 MAT A=ZER(2,3)
0030 MAT PRINT A;
0040 STØP

     0     0     0   } Résultat de
     0     0     0   } l'exécution
```

7.4.4. Matrice unité.

L'instruction s'écrit : MAT A = IDN ou
MAT A = IDN(I, I)

Dans cette instruction :

- A est le nom de la matrice,
- les indices I sont les dimensions de la matrice (quand ils sont précisés dans l'instruction) qui doit être carrée.

Cette opération affecte la valeur 1 aux éléments de la diagonale principale et 0 aux autres éléments.

Exemple :

```
0010 DIM A(8,8)
0020 LET N=1
0030 MAT A=IDN(2*N+1,2*N+1)
0040 MAT PRINT A;
0050 STØP

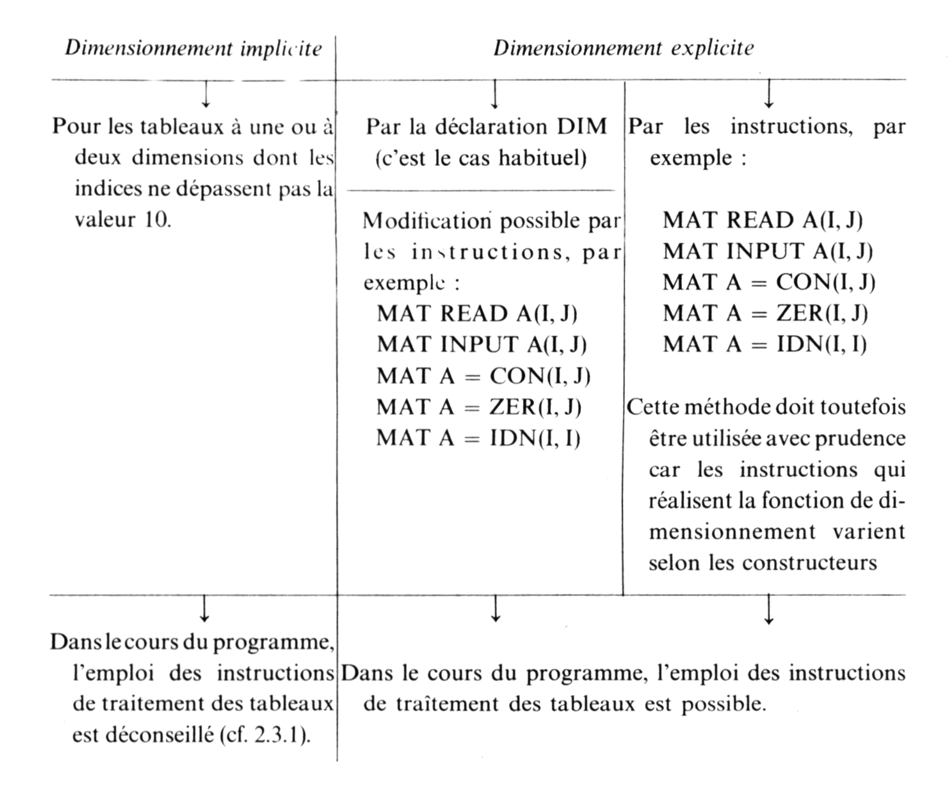
     1     0     0   }
     0     1     0   } Résultat de l'exécution
     0     0     1   }
```

L'instruction IDN a permis de redimensionner la matrice (ce point est développé au paragraphe qui suit).

7.5. Les différentes possibilités pour dimensionner les tableaux.

Le dimensionnement des tableaux peut être réalisé :
— par une déclaration DIM,
— par certaines instructions de traitement des tableaux.

La figure 7.3 regroupe les diverses possibilités offertes à l'utilisateur, montre à quel endroit du programme elles peuvent être employées, et les conséquences qui en résultent.



Dimensionnement implicite	*Dimensionnement explicite*	
Pour les tableaux à une ou à deux dimensions dont les indices ne dépassent pas la valeur 10.	Par la déclaration DIM (c'est le cas habituel) Modification possible par les instructions, par exemple : MAT READ A(I, J) MAT INPUT A(I, J) MAT A = CON(I, J) MAT A = ZER(I, J) MAT A = IDN(I, I)	Par les instructions, par exemple : MAT READ A(I, J) MAT INPUT A(I, J) MAT A = CON(I, J) MAT A = ZER(I, J) MAT A = IDN(I, I) Cette méthode doit toutefois être utilisée avec prudence car les instructions qui réalisent la fonction de dimensionnement varient selon les constructeurs
Dans le cours du programme, l'emploi des instructions de traitement des tableaux est déconseillé (cf. 2.3.1).	Dans le cours du programme, l'emploi des instructions de traîtement des tableaux est possible.	

FIG. 7.3.

Il convient également de rappeler que les indices I et J :

— peuvent être des constantes, des variables, ou des expressions arithmétiques, définies préalablement à l'instruction, qui sont tronquées si elles ne sont pas entières (cf. 2.3),

— peuvent être négatifs pour certains systèmes (cf. 2.1.2).

L'exemple qui suit présente deux combinaisons de dimensionnement :

— les dimensions du tableau A sont fixées par l'instruction READ seule;

— les dimensions du tableau B sont déclarées par DIM puis modifiées par l'instruction « CONSTANTE ».

```
0010 DIM B(20,20)
0020 READ M,N
0030 MAT READ A(2,3)
0040 MAT B=CØN(M+1.5,N+2)
0050 PRINT "TABLEAU A"
0060 MAT PRINT A;
0070 PRINT "TABLEAU B"
0080 MAT PRINT B;
0090 STØP
0100 DATA 2,3
0110 DATA 1,2,3,4,5,6

TABLEAU A
    1       2       3
    4       5       6

TABLEAU B
    1       1       1       1       1
    1       1       1       1       1
    1       1       1       1       1
```

Résultat de l'exécution

7.6. *Exercices.*

7.6.1. Résolution d'un système linéaire.

Tout système d'équations linéaires de la forme :

$$a_0^0 x_0 + a_0^1 x_1 + \ldots\ldots + a_0^n x_n = b_0$$

$$a_1^0 x_0 + a_1^1 x_1 + \ldots\ldots + a_1^n x_n = b_1$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$a_n^0 x_0 + a_n^1 x_1 + \ldots\ldots + a_n^n \ x_n + b_n$$

peut se représenter sous forme matricielle en introduisant des matrices :

$$A = \begin{bmatrix} a_0^0 \, a_0^1 \, \ldots\ldots \, a_0^n \\ a_1^0 \, a_1^1 \, \ldots\ldots \, a_1^n \\ \ldots\ldots\ldots\ldots \\ a_n^0 \, a_n^1 \, \ldots\ldots \, a_n^n \end{bmatrix} \quad X = \begin{bmatrix} x_0 \\ x_1 \\ \ldots\ldots \\ x_n \end{bmatrix} \quad B = \begin{bmatrix} b_0 \\ b_1 \\ \ldots\ldots \\ b_n \end{bmatrix}$$

On a alors $A\,X = B$

d'où $A^{-1}\,A\,X = A^{-1}\,B$

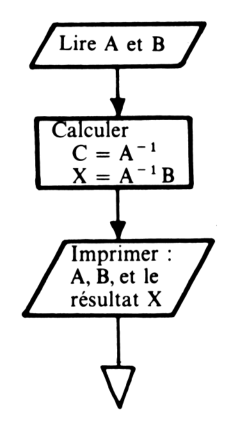
et la solution est : $X = A^{-1}\,B$.

La solution nous est fournie par l'application de l'instruction inversion de matrices (l'exercice 12.1.8 résout également ce problème sans utiliser les instructions de calcul matriciel).

Soit par exemple le système :

$$\begin{aligned} 6x_1 \qquad\quad - 4x_3 &= -10 \\ 3x_1 - 8x_2 + 2x_3 &= 39 \\ 5x_1 + 7x_2 - x_3 &= -6 \end{aligned}$$

Sa transposition sous forme de programme est immédiate.



Les valeurs numériques sont les suivantes :

$$A = \begin{bmatrix} 6 & 0 & -4 \\ 3 & -8 & 2 \\ 5 & 7 & -1 \end{bmatrix}$$

$$B = \begin{bmatrix} -10 \\ 39 \\ -6 \end{bmatrix}$$

FIG. 7.4.

```
0010 DIM A(3,3),B(3,1),C(3,3),X(3,1)
0020 MAT READ A
0030 MAT READ B
0040 MAT C=INV(A)
0050 MAT X=C*B
0060 PRINT "    MATRICE A"
0070 MAT PRINT A;
0080 PRINT "    MATRICE B"
0090 MAT PRINT B;
0100 PRINT "    MATRICE DES RESULTATS"
0110 MAT PRINT X;
0120 STØP
0130 DATA 6,0,-4,3,-8,2,5,7,-1
0140 DATA -10,39,-6
```

```
    MATRICE A
    6      0 -    4
    3 -    8      2
    5      7 -    1

    MATRICE B
-   10
   39
-    6

    MATRICE DES RESULTATS
 .300000E 01
-.200000E 01
 .700000E 01
```

Résultat de l'exécution

Le résultat est bien : $x_1 = 3$, $x_2 = -2$, $x_3 = 7$.

7.6.2. Régressions linéaires : le problème de l'ajustement des données.

Dans de nombreuses applications, il est intéressant de rechercher une équation liant entre eux les résultats de plusieurs observations expérimentales.

Pour ce faire, on se donne *a priori* la forme de l'équation (on l'appelle aussi le « modèle »).

On se fixe également des « critères » et, en fonction de ceux-ci, on ajuste au mieux le modèle à la réalité observée.

Pratiquement, le problème consiste alors à déterminer les coefficients de l'équation de telle sorte que l'écart entre la « prédiction » (c'est-à-dire les résultats calculés) et les résultats expérimentaux, soit minimum.

Très souvent, le modèle choisi est linéaire : les instructions matricielles permettent de le traiter élégamment.

7.6.2.1. Régression linéaire simple et critère des moindres carrés.

Un résultat d'expérience y est fonction d'une seule variable x.

Nous choisissons d'appliquer un modèle linéaire de la forme :

$$y = ax + b.$$

Nous décidons également d'utiliser le « critère des moindres carrés » qui se définit comme suit : déterminer a et b de telle façon que :

$\sum_{i=1}^{i=N} [y_i - (ax_i + b)]^2$ soit minimum, les indices i étant les numéros des expériences.

En différenciant par rapport à a :

$$\sum 2[y_i - (ax_i + b)](-x_i) = 0, \quad \text{ou :} \quad -\sum x_i y_i + a\sum x_i^2 + b\sum x_i = 0$$

En différenciant par rapport à b :

$$\sum 2[y_i - (ax_i + b)](-1) = 0, \quad \text{ou :} \quad \sum y_i - a\sum x_i - Nb = 0.$$

En résolvant ce système par rapport à a et b on obtient :

$$a = \frac{\sum x_i y_i - \dfrac{\sum x_i \sum y_i}{N}}{\sum x_i^2 - \dfrac{(\sum x_i)^2}{N}} \quad \text{et} \quad b = \frac{\sum y_i - a\sum x_i}{N}$$

Le programme ci-après est l'application directe de ces équations. On notera l'introduction des variables auxiliaires :

$$X1 = \sum x_i \qquad X2 = \sum x_i^2$$
$$Y1 = \sum y_i \qquad Z = \sum x_i y_i$$

Exemple : les résultats expérimentaux sont les suivants :

Expérience	1	2	3	4	5	6	7	8	N = 9
x	32	50	70	80	100	120	140	150	170
y	80	89	91	98	99	106	108	116	115

FIG. 7.5.

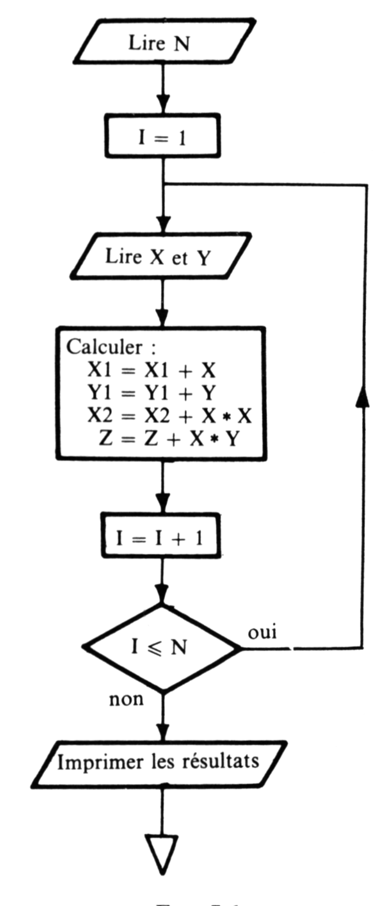


FIG. 7.6.

```
0010 LET X1=0
0020 LET X2=0
0030 LET Y1=0
0040 LET Z=0
0050 READ N
0060 FØR I=1 TØ N
0070      READ X,Y
0080      LET X1=X1+X
0090      LET Y1=Y1+Y
0100      LET X2=X2+X*X
0110      LET Z=Z+X*Y
0120 NEXT I
0130 LET A=(Z-X1*Y1/N)/(X2-X1*X1/N)
0140 PRINT "     NØMBRE DE PØINTS =",N
0150 PRINT "     PENTE DE LA DRØITE =",A
0160 PRINT "     ØRDØNNEE A L'ØRIGINE =",Y1/N-A*X1/N
0170 STØP
0190 DATA 9
0200 DATA 32,80,50,89,70,91
0210 DATA 80,98,100,99,120,106
0220 DATA 140,108,150,116,170,115
```

```
NØMBRE DE PØINTS =                9
PENTE DE LA DRØITE = .253107E 00
ØRDØNNEE A L'ØRIGINE = .745740E 02
```

} Résultat de l'exécution

7.6.2.2. Régression linéaire multiple.

On peut également être conduit à rechercher un modèle linéaire lorsque y est fonction de p variables : $x_1, x_2, ..., x_p$.

Dans ce cas, étant donné n expériences ($n > p$ pour obtenir une analyse correcte), on peut écrire :

$$y_1 = x_{11}a_1 + x_{12}a_2 + ... + x_{1p}a_p$$
$$y_2 = x_{21}a_1 + x_{22}a_2 + ... + x_{2p}a_p$$
$$..........$$
$$..........$$
$$y_n = x_{n1}a_1 + x_{n2}a_2 \qquad + x_{np}a_p$$

Ce système devient, sous forme matricielle :

$$Y = X.A.$$

Cette équation ne peut pas être directement résolue pour A, car X a plus de lignes que de colonnes.

L'application de la méthode des moindres carrés à ce système conduit à multiplier les deux membres de l'équation par la matrice transposée de X, notée V.

$$V.Y = V.X.A.$$

En multipliant les deux membres par $(V.X)^{-1}$ on obtient :

$$(V.X)^{-1}V.Y = (V.X)^{-1}V.X.A.$$

Alors : $$A = (V . X)^{-1} V . Y.$$

Le calcul de A s'effectue très aisément à l'aide des instructions matricielles, comme le montre l'exemple qui suit.

Exemple :

On décide d'appliquer un modèle linéaire à la fonction y qui dépend de quatre variables x :

$$y = x_1 a_1 + x_2 a_2 + x_3 a_3 + x_4 a_4.$$

Les résultats de cinq expériences sont consignés dans le tableau ci-dessous :

Numéro d'expérience	x_1	x_2	x_3	x_4	y
1	1,89	3,52	2,71	4,59	107,914
2	3,92	7,09	9,18	8,51	223,076
3	6,38	3,21	7,38	4,41	157,361
4	3,21	1,09	2,91	7,11	134,566
5	5,20	2,88	3,41	8,29	181,154

FIG. 7.7.

Toutes ces quantités sont exploitées sous forme matricielle. En particulier, le programme calcule la matrice A dont les éléments a_1 à a_4 s'ajustent au mieux aux résultats des cinq expériences.

Les coefficients a une fois déterminés, le programme calcule la « prédiction », c'est-à-dire les valeurs de y qui sont obtenues à chaque expérience en appliquant la formule :

$$K = XA.$$

Le calcul de la variance :

$$S^2 = \left[\sum_{i=1}^{i=N} (y_i - k_i)^2 \right] / (n - p)$$

renseigne sur la validité de l'ajustement.

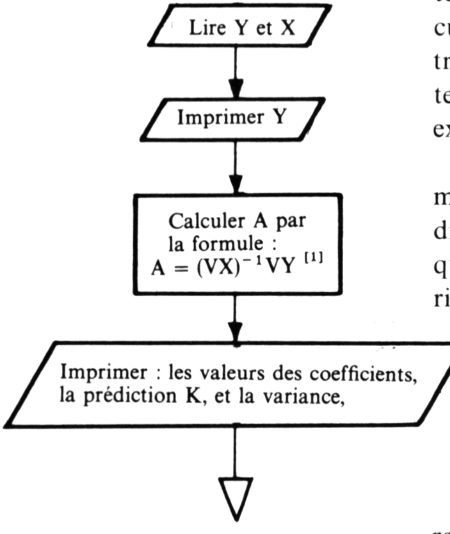


FIG. 7.8.

(1) Les matrices auxiliaires V, W, T et Z sont introduites dans le programme dans le seul but de ne pas faire apparaître le même nom de matrice des deux côtés du signe « égal ».

```
0010 DIM Y(20,1),K(20,1),X(20,19),V(19,20)
0020 DIM W(19,19),T(19,19),Z(19,19),A(19,1)
0030 READ N,P
0040 MAT Y=ZER(N,1)
0050 MAT K=ZER(N,1)
0060 MAT X=ZER(N,P)
0070 MAT V=ZER(P,N)
0080 MAT W=ZER(P,P)
0090 MAT T=ZER(P,P)
0100 MAT Z=ZER(P,N)
0110 MAT A=ZER(P,1)
0120 MAT READ Y,X
0123 PRINT "   VALEURS DE Y MESUREES"
0124 MAT PRINT Y
0130 MAT V=TRN(X)
0140 MAT W=V*X
0150 MAT T=INV(W)
0160 MAT Z=T*V
0170 MAT A=Z*Y
0180 PRINT "   VALEUR DES CØEFFICIENTS"
0190 MAT PRINT A
0200 MAT K=X*A
0210 PRINT "   PREDICTIØN PØUR Y"
0220 MAT PRINT K
0230 LET S2=0
0240 FØR I=1 TØ N
0250      LET S2=S2+(Y(I,1)-K(I,1))↑2
0260 NEXT I
0270 PRINT "   VARIANCE =",S2/(N-P)
0280 STØP
0290 DATA 5,4
0300 DATA 107.914,223.076,157,361,134.566,181.154   <-- Valeurs de Y
0310 DATA 1.89,3.52,2.71,4.59  )
0320 DATA 3.92,7.09,9.18,8.51  )
0330 DATA 6.38,3.21,7.38,4.41  }  <-- 5 jeux de valeurs pour X
0340 DATA 3.21,1.09,2.91,7.11  )
0350 DATA 5.20,2.88,3.41,8.29  )
```

```
  VALEURS DE Y MESUREES
.107914E 03
.223076E 03
.157361E 03
.134566E 03
.181154E 03

  VALEUR DES CØEFFICIENTS
.832733E 01
.710785E 01
.346636E 01
.126922E 02

  PREDICTIØN PØUR Y
.108409E 03
.222870E 03
.157499E 03
.134807E 03
.180811E 03

  VARIANCE = .482420E 00
```

Résultat de l'exécution

CHAPITRE 8

LES VARIABLES CARACTÈRES ET LEUR UTILISATION

8.0. Introduction.

Les variables numériques utilisées précédemment permettent d'effectuer des calculs et d'éditer des valeurs numériques.

Par contre elles ne permettent pas de traiter des données alphanumériques. Pour agrandir le champ d'application du BASIC (notamment en gestion) il a été adjoint au BASIC initial la notion de « variable et de constante caractères ».

8.1. Les opérations sur chaînes de caractères.

Il y a lieu de dissocier la notion de « chaîne de caractères » telle qu'on la rencontre en mathématiques de celle, plus restrictive, du BASIC. Pour cela nous allons donner quelques précisions.

8.1.1. Chaîne de caractères.

Le vocable « chaîne de caractères » a été défini par les mathématiciens pour leur permettre de mieux étudier certains problèmes de mathématiques non numériques tels que étude de linguistique des langues naturelles, analyse syntaxique des langages de programmation, etc.

En BASIC, la notion de chaîne de caractères désigne simplement un ensemble de caractères placés les uns à la suite des autres. Par exemple le libellé :

BELLE MARQUISE

est une chaîne de caractères comportant quatorze caractères.

Un caractère peut être n'importe quel élément de l'alphabet disponible sur l'ordinateur utilisé exception faite toutefois pour le caractère ' (quote) qui sert de séparateur.

En BASIC la longueur d'une chaîne de caractères est limitée à une longueur de 15 à 18 caractères pour de nombreux systèmes (cf. paragraphe 2.3.2).

Seuls des systèmes très évolués (XDS 940 - CDC - Pdp 11 - MARK II) admettent des longueurs supérieures.

Cette limitation est due au fait que la capacité mémoire centrale disponible pour ce stockage est limitée et aux contraintes imposées par le compilateur BASIC.

8.1.2. Opérations disponibles sur des chaînes de caractères.

On peut définir différentes opérations portant des chaînes de caractères. Ces opérations sont d'autant plus nombreuses que le langage est destiné à un tel traitement. Cependant on peut retenir trois opérations fondamentales qui sont :

— l'affectation,
— la comparaison,
— la concaténation.

Opération d'affectation : cette opération consiste à ranger le contenu d'une chaîne dans une autre.

Opération de comparaison : il faut pouvoir comparer le contenu de deux chaînes de caractères.

Deux chaînes seront dites « égales » si elles contiennent la même chaîne de caractères. En BASIC on pourra étendre les possibilités de comparaison.

Opération de concaténation : cette opération permet de mettre bout à bout deux chaînes de caractères afin d'en constituer une troisième.

Exemple :

" ABC " concaténée avec " DEFGH "

donnera

" ABCDEFGH "

cette opération n'est pas disponible sur tous les systèmes BASIC.

L'opérateur de concaténation utilisé est le signe +, exemple :

```
100 LET X$ = " BELLE "
110 LET X$ = X$ + "    MARQUISE "
```

la chaîne X$ prendra la valeur " BELLE MARQUISE ".

Opérations diverses : des possibilités d'extraction de sous chaîne à partir d'une chaîne donnée, de détermination de la longueur d'une chaîne, etc. sont mentionnées dans le tableau de la figure 8.2.

8.2. L'instruction d'affectation pour variables caractères.

La forme originelle de cette instruction était

LET VAR_1, VAR_2, VAR_n = chaîne de caractères

dans laquelle toutes les variables doivent être des variables caractères.

Exemples :

LET Y$ = X$ = " ABC "
LET Y$ = A$
LET V$(1) = " ABCD ".

Maintenant la plupart des systèmes acceptent des formes plus évoluées de cette instruction permettant d'utiliser l'opération de concaténation (cf. 8.1) et les fonctions de chaînes.

8.3. L'instruction de comparaison sur chaînes de caractères.

La représentation interne des caractères permet d'établir une « relation d'ordre » entre les différents caractères de l'alphabet et ainsi d'étendre les possibilités de comparaison en BASIC.

Cette représentation interne dépend du système utilisé. Le code ASCII très répandu classe les caractères dans l'ordre suivant (fig. 8.1) :

A partir de ce classement « alphabétique » nous prendrons la définition suivante :

Définition : Une chaîne de caractères A$ est *plus petite* qu'une chaîne B$ si dans un classement par ordre alphabétique, la chaîne A$ se trouve placée avant la chaîne B$.

Exemple : " CHAINE " < " CHAMBORD "
" CHAMBORD " < " CHAT "

Le symbole « plus petit que » sera le même que celui utilisé pour des variables numériques.

Caractères	*Valeur en décimal*	*Caractères*	*Valeur en décimal*
^	32	@	64
!	33	A	65
"	34	B	66
#	35	C	67
$	36	D	68
%	37	E	69
&	38	F	70
'	39	G	71
(	40	H	72
)	41	I	73
*	42	J	74
+	43	K	75
,	44	L	76
–	45	M	77
.	46	N	78
/	47	O	79
0	48	P	80
1	49	Q	81
2	50	R	82
3	51	S	83
4	52	T	84
5	53	U	85
6	54	V	86
7	55	W	87
8	56	X	88
9	57	Y	89
:	58	Z	90
;	59	[	91
<	60	\	92
=	61	]	93
>	62	↑	94
?	63	←	95

Ceci signifie par exemple que "/" < "2" < ":" < "A"

FIG. 8.1.

Cette définition permet de disposer des mêmes opérateurs de comparaison que pour les variables numériques :

< plus petit que
< = plus petit ou égal
> plus grand que
> = plus grand ou égal
= égal
< > différent de

L'instruction de comparaison aura donc l'allure suivante :

IF ⟨expression de chaîne⟩ ⟨opérateur de comparaison⟩ ⟨expression de chaîne⟩ THEN ⟨numéro de ligne⟩

Exemple :

IF X$(I) < X$(J) THEN 100
IF X$(I) = " ABC " THEN 50

L'intérêt de telles instructions apparaîtra notamment à propos des exercices de TRI.

8.4. Les instructions de lecture pour variables caractères.

Les listes de variables d'entrée des instructions READ et INPUT peuvent contenir des variables caractères simples ou des éléments de tableaux caractères.

La présentation d'une chaîne de caractères dans une instruction DATA devra être séparée des autres données par un séparateur (virgule) mais pas obligatoirement encadré de deux guillemets sauf si cette chaîne contient des caractères spéciaux (virgule, point virgule, blanc en début ou fin de chaîne).

Exemples :

```
READ X,A$,D$(I)
DATA 10,Z0E,"BLUFF"
```

Correct mais les guillemets ne sont pas obligatoires.

```
READ X,A$,D$(I)
DATA 10,Z0E,"  BLUFF"
```

Correct mais les guillemets sont ici pour indiquer que la chaîne commence par deux blancs

```
READ X,A$,B$(I)
DATA 10,SEPARATEUR,EST UTILE
```

incorrect car cette chaîne contient une virgule. Il faut écrire :

```
DATA 10,SEPARATEUR,",EST UTILE"
```

8.5. Les instructions PRINT et PRINT USING pour chaînes de caractères.

8.5.1. L'instruction PRINT pour chaînes de caractères.

La liste d'une instruction PRINT peut contenir :
— des expressions arithmétiques,
— des chaînes de caractères.

— Si une chaîne est suivie d'une virgule elle est éditée dans une zone dont la longueur est de 15 caractères (ou 18 caractères pour IBM), et est cadrée à gauche de la zone allouée.

— Si elle est suivie d'un point virgule, la variable suivante éventuelle sera éditée sans espace intermédiaire.

— Si la longueur de la chaîne est inférieure à la longueur de la zone, l'impression se fait avec cadrage à gauche et le reste de la zone est complété par des caractères « blancs ».

— Si la longueur de la chaîne est supérieure à la longueur de la zone, les premiers caractères de la chaîne sont imprimés et il y a troncature lorsque la zone est remplie.

8.5.2. L'instruction PRINT USING pour chaînes de caractères.

Une instruction PRINT USING peut contenir des variables caractères qui seront alors éditées en tenant compte des règles générales suivantes valables sur plusieurs systèmes :

— Les caractères de la chaîne seront (sauf cas particulier) cadrés à gauche.

— Si la zone allouée a une longueur supérieure à ce qui est nécessaire, la zone est complétée à droite par des blancs.

— Si la longueur de la zone est trop courte pour recevoir la chaîne tout entière, celle-ci est tronquée sur sa partie droite.

Chaque système présente des particularités notamment en ce qui concerne les instructions PRINT USING et IMAGE, particularités trop nombreuses pour figurer dans cet ouvrage.

Il y a lieu de noter cependant que certains systèmes (CDC, XDS 940) ne disposent pas de cette instruction mais des instructions PRINT IN IMAGE et PRINT IN FORM qui est inspirée du FORTRAN.

8.6. Les fonctions standard pour chaînes de caractères.

Ces fonctions ont été ajoutées peu à peu au BASIC original et leur nom et leur interprétation diffèrent beaucoup d'un système à l'autre.

Les principales fonctions sont indiquées dans le tableau figure 8.2.

Certaines fonctions liées au système sont des fonctions chaînes particulières.

Désignations	*Noms*	*Commentaires*
S/chaîne gauche de A$, longueur N	LEFT(A$, N)	Donne une sous chaîne contenant les N premiers caractères de A$
S/chaîne droite de A$. longueur N	RIGHT(A$, N)	Donne une sous chaîne contenant les N derniers caractères de A$
Longueur de A$	LEN(A$)	Donne un nombre entier égal au nombre de caractères de la chaîne A$
Position de B$ dans A$	INDEX(A$, B$)	Donne la position de B$ dans A$ Existe sur les systèmes XDS 940 et CDC
Recherche de B$ dans A$	INSTR(N, A$, B$)	Indique pour résultat 1 ou 0 si la chaîne B$ est contenue ou non dans la chaîne A$ à partir du N^ieme caractère de A$. Existe sur le système Pdp 11.
S/chaîne extraite de A$	SUBSTR(A$, N1, N2) ou MID $(A$, N1, N2)	Donne une sous chaîne extraite de A$ à partir du caractère N1 et d'une longueur N2.
« Blanc » de longueur N	SPACE(N)	Donne une chaîne contenant N « Blancs »
Conversion ASCII binaire	VAL(A$)	Donne un nombre contenant la valeur numérique de la chaîne A$ exprimée selon le code ASCII
Conversion binaire ASCII	STR(N)	Donne une chaîne de caractères contenant la valeur de N exprimée en code ASCII

FIG. 8.2.

8.7. Exercices.

A partir des chaînes de caractères suivantes : « BELLE MARQUISE », « VOS BEAUX YEUX », « ME FONT », « MOURIR », « D'AMOUR » (1).

Construire différentes phrases susceptibles d'être formées. On procédera de la façon suivante :

Ces chaînes seront rangées dans les éléments d'une variable caractère indicée A$ (comportant cinq éléments). Le programme demandera à lire cinq chiffres de l'ensemble (1, 2, 3, 4, 5) indiquant dans quel ordre il faut

(1) Le « Bourgeois gentilhomme », Molière, acte II, scène IV.

« concaténer » ces chaînes en vue de constituer une nouvelle chaîne B$ qui sera alors imprimée.

Ici on fera un programme qui « boucle sur lui-même c'est-à-dire qui ne peut pas s'arrêter de lui-même et qui demande sans cesse une nouvelle séquence de chiffres.

Exemple : 5, 2, 3, 4, 1 doit provoquer l'impression de

D'AMOUR VOS BEAUX YEUX ME FONT MOURIR
BELLE MARQUISE

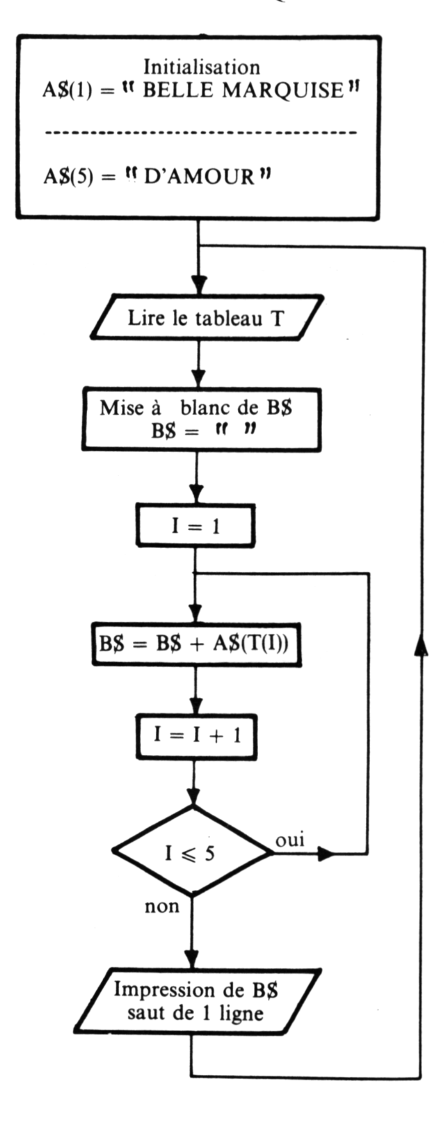


FIG. 8.3.

A partir de l'organigramme précédent, l'écriture de la séquence suivante ne présente aucune difficulté :

```
10 DIM A$(5),T(5)
20 LET A$(1)=" BELLE MARQUISE"
30 LET A$(2)=" VOS BEAUX YEUX"
40 LET A$(3)=" ME FONT"
50 LET A$(4)=" MOURIR"
60 LET A$(5)=" D'AMOUR"
65 MAT INPUT T
66 LET B$=""
70 FOR I=1 TO 5
80 LET B$=B$+A$(T(I))
90 NEXT I
100 PRINT B$
101 PRINT
105 GO TO 65
110 END
```

résultat à l'exécution :

```
? 1,2,3,4,5
 BELLE MARQUISE VOS BEAUX YEUX ME FONT MOURIR D'AMOUR

? 1,3,2,4,5
 BELLE MARQUISE ME FONT VOS BEAUX YEUX MOURIR D'AMOUR

? 1,2,3,5,4
 BELLE MARQUISE VOS BEAUX YEUX ME FONT D'AMOUR MOURIR

? 2,3,4,5,1
 VOS BEAUX YEUX ME FONT MOURIR D'AMOUR BELLE MARQUISE

? 1,5,3,4,2
 BELLE MARQUISE D'AMOUR ME FONT MOURIR VOS BEAUX YEUX
```

CHAPITRE 9

INSTRUCTIONS, FONCTIONS ET SUBROUTINES LIÉES AU SYSTÈME D'EXPLOITATION

9.0. Introduction.

Le langage BASIC est rendu disponible sur un ordinateur donné à partir de différents dispositifs : sur le plan matériel soit à partir de la machine à écrire de l'ordinateur, soit à partir d'un terminal de time sharing et sur le plan software via un « système d'exploitation » (on dit aussi moniteur) plus ou moins sophistiqué selon l'importance du système.

Le système d'exploitation a en général une influence importante sur les entrées/sorties, et en particulier sur les fichiers. Il a également une influence sur certaines fonctions standard qui sont liées au système et sur certaines particularités accessibles en BASIC.

9.1. Fonctions standard liées au système.

Ces fonctions sont évidemment variables d'un système à l'autre et ce paragraphe présente l'esprit dans lequel ces fonctions ont été créées et ce à quoi elles peuvent servir.

9.1.1. Mesure du temps unité centrale consommé depuis le début de l'exécution d'un programme.

Cette fonction peut être utilisée pour comparer la rapidité de calcul de méthodes mathématiques. Par exemple pour résoudre un problème nous disposons de deux algorithmes et désirons déterminer le plus rapide, pour cela nous aurons deux programmes dont l'allure générale est donnée par l'organigramme de la figure 9.1.

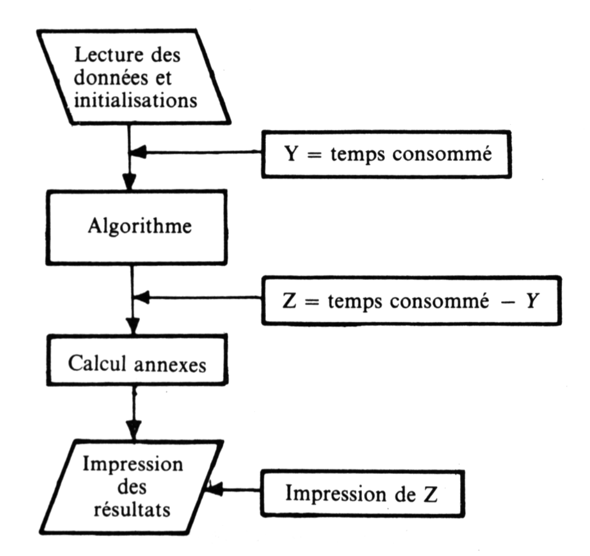


FIG. 9.1.

En encadrant la séquence d'instructions relative à l'algorithme par des mesures du temps consommé, on peut obtenir le temps unité centrale consommé par l'exécution de l'algorithme proprement dit. Il est ainsi possible de comparer plusieurs algorithmes relatifs à un même problème.

Cette mesure se fait au moyen d'une fonction dont la forme dépend du système :

SYSTEMES PHILIPS et SIEMENS

Y = TIM (argument) unité = seconde d'unité centrale

l'argument est nécessaire pour des raisons de syntaxe mais sans signification.

SYSTEME HONEYWELL-BULL (MARK II)

Y = TIM unité = seconde unité centrale

SYSTEME Pdp 11

Y = TIM (1) unité = 0,1 seconde unité centrale.

9.1.2. Fonctions date et heure.

Ces fonctions prennent une forme très diversifiées selon les systèmes. Elles donnent respectivement la date calendrier et l'heure.

SIGMA 5/7 Y = YER (*x*) donne l'année en cours
Z = DAY (*x*) donne le jour calendrier
V = TIM (*x*) donne l'heure depuis minuit

la présentation du résultat dépend de la valeur du paramètre X.

HP 2 000 C TIM (0) donne le nombre de minutes écoulées depuis le début de l'heure (de 0 à 59)
TIM (1) donne l'heure (de 0 à 23)
TIM (2) donne le jour (de 0 à 366)
TIM (3) donne l'année (de 0 à 99)

Pdp 11 TIM (0) donne le temps en secondes après minuit.
TIM (1) donne le temps unité centrale consommé depuis le début de l'exécution du programme
unité = 0,1 seconde

TIM (2) donne le temps de connexion de l'utilisateur depuis qu'il est entré en contact avec le système.

9.1.3. Fonctions liées à l'état du système.

Certains systèmes disposent de fonctions telles que WAIT et SLEEP qui suspendent l'exécution pendant une durée déterminée. L'intérêt réside surtout dans le fait que l'on peut demander ainsi à attendre qu'une entrée soit effectuée avant qu'un certain temps ne soit écoulée.

La fonction TEL permet sur certains systèmes de connaître l'état du terminal de l'utilisateur.

9.1.4. Entrées/sorties industrielles.

Certains ordinateurs de contrôle de processus (VARIAN par exemple) permettent dans une certaine mesure de programmer des applications de contrôle de processus en BASIC. Pour cela le BASIC a été dotée de sous-programmes permettant d'effectuer des entrées/sorties relatives à des dispositifs de mesures analogiques ou logiques. En outre des entrées/sorties particulières permettent d'exploiter une console graphique.

Ces fonctions sont appelées au moyen de l'instruction CALL (cf. chap. 11).

9.2. Le chaînage des programmes.

Il arrive qu'un programme soit trop encombrant pour pouvoir être exécuté directement en mémoire centrale. Dans ce cas il faut alors scinder le programme en plusieurs tronçons afin d'utiliser des techniques « d'*Overlay* ».

Sur de nombreux systèmes, ceci est rendu possible, en BASIC, au moyen de l'instruction CHAIN dont la syntaxe de l'allure suivante :

CHAIN nom de programme

ou

CHAIN variable caractère
la variable caractère doit contenir le nom du programme à exécuter.

Cette instruction permet donc d' « enchaîner » consécutivement et sans intervention de l'utilisateur plusieurs programmes, ou en fait différents modules résultant du découpage d'un programme unique trop volumineux.

Ce chaînage pose le problème suivant : les différents modules chargés successivement en mémoire centrale travaillent sur le plan théorique à partir de données communes. Comment va-t-on gérer la mémoire pour permettre l'utilisation de données communes?

Le BASIC de façon classique propose aux utilisateurs, avant d'appeler un autre module, d'écrire les données à réutiliser sur disques. Le module appelé en second doit alors relire ces données à partir du disque.

Cette méthode est un peu archaïque et certains constructeurs comme Hewlett-Packard, mettent à la disposition de l'utilisateur l'instruction COM (inspirée de l'instruction COMMON du FORTRAN) qui permet de réserver en mémoire centrale une zone dans laquelle seront placées les variables communes à différents modules.

9.3. Le mode calculateur de bureau en basic.

Un ordinateur dispose d'une puissance suffisante pour pouvoir simuler une machine à calculer électronique de bureau. Parmi les méthodes de simulation, l'une d'elles consiste à étendre le langage BASIC de façon à ce que chaque instruction puisse elle-même être exécutée. De telles instructions n'ont pas de numéro de ligne et les possibilités sont limitées essentiellement aux instructions PRINT.

Exemple : 1) PRINT 3, 3 * CØS (2) * EXP(−0,5)
2) PRINT X, X ↑ 3 + 5 FOR X = 1 TO 10

Dès l'instant où une telle ligne est frappée, l'ordinateur exécute l'instruction.

Ce mode constitue souvent une mauvaise utilisation de l'ordinateur.

CHAPITRE 10

LES INSTRUCTIONS DE TRAITEMENT DES FICHIERS

10.0. Introduction.

Les entrées-sorties présentées au chapitre 5 et dont certains compléments sont mentionnés au chapitre 11 concernent essentiellement les échanges d'information entre l'ordinateur et l'utilisateur, ce dernier communiquant avec l'organe de traitement au moyen d'un « terminal » dont le débit maximum est de l'ordre de 10 à 30 caractères par seconde.

Les ordinateurs disposent par ailleurs de « mémoires auxiliaires » (disques et bandes magnétiques) dont la capacité de stockage est très supérieure à celle des « mémoires centrales », qui permettent de ranger à titre temporaire ou permanent une bonne quantité d'informations à un prix raisonnable.

Compte tenu de l'importance des « fichiers » pour les applications scientifiques et les applications de gestion, ce chapitre :

- présente les notions générales relatives aux fichiers, valables pour différents langages de programmation ;
- indique la forme des instructions (1) les plus courantes ;
- signale quelques extensions intéressantes du langage, et
- propose un exercice d'application.

10.1. Notions générales sur les fichiers.

L'une des propriétés des bandes magnétiques et de certains disques magnétiques est leur « amovibilité » : après avoir écrit sur ces mémoires,

(1) La forme des instructions n'est pas la même pour les différents compilateurs, mais les fonctions réalisées sont similaires.

l'utilisateur peut les démonter de l'unité de lecture-écriture et les ranger, afin de disposer ultérieurement des informations qui y ont été stockées.

Les caractéristiques physiques de ces mémoires entraînent pour l'utilisateur des contraintes qui apparaissent dans une certaine mesure au niveau des instructions d'entrée-sortie.

Ces caractéristiques générales sont les suivantes :

- échange d'informations par bloc et non par caractère,
- accès « séquentiel » seulement, pour les bandes magnétiques,
- possibilité d'accès « sélectif » pour les disques magnétiques.

Il s'en suit que les fichiers pourront avoir des structures différentes suivant le type de mémoire utilisé.

10.1.1. Constitution des fichiers.

Le langage courant désigne par le terme de « fichier » un ensemble de « fiches ». Celles-ci sont par exemple des feuilles cartonnées dont la mise en page (ou format) est identique pour tous les éléments de la collection.

La transposition au domaine du traitement de l'information est immédiate.

A la notion de fiche correspond la notion *d'article.*

L'article est un ensemble de *données* qui se rapportent au même objet.

Un fichier est donc un ensemble d'articles enregistrés sur un support selon un format identique pour tous les éléments de l'ensemble.

Par exemple, pour définir un stock de pièces détachées, on peut constituer un fichier dont chaque article représente une pièce du stock et comporte les données suivantes :

— première donnée : numéro codifié de la pièce,
— deuxième donnée : sa désignation en langage clair,
— troisième donnée : son prix,
— quatrième donnée : la quantité restant en stock.

Sur cet exemple, on voit que la désignation en langage clair peut-être très longue si l'on veut faire intervenir toutes les qualités d'une pièce (par exemple : la couleur pour des tissus). Cette donnée se prête donc très mal aux traitements en ordinateur.

Le numéro codifié, lui, est plus concis.

C'est aussi la seule donnée sur laquelle on puisse définir une relation bi-univoque avec l'article : de ce fait, il est employé pour véritablement individualiser l'article; on l'appelle *identificateur* ou *indicatif* (le vocable « clé » est aussi employé).

L'indicatif est aussi un moyen pour ranger l'article dans le fichier, et ultérieurement pour le retrouver.

Le paragraphe 10.1.4 précise la nature des indicatifs admise par le langage BASIC.

10.1.2. Les supports de fichiers.

Hormis le cas particulier où ils sont résidents en mémoire centrale, les fichiers sont placés dans les mémoires auxiliaires de l'ordinateur.

Le langage BASIC comporte donc des instructions d'entrée-sortie qui permettent les échanges d'information entre ces deux types de mémoires. Ainsi, un fichier de cartes perforées est « lu » par un programme comportant une suite d'instructions d'entrée.

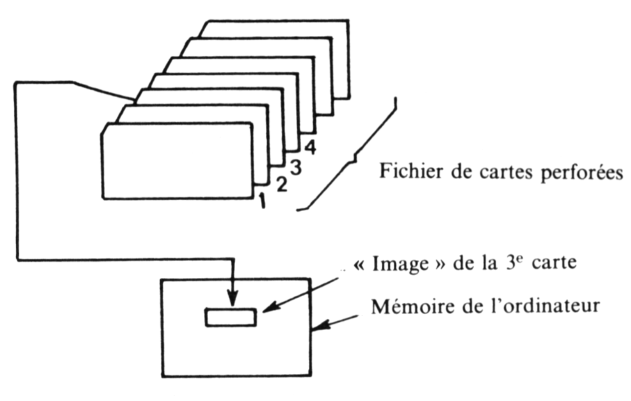


Fig. 10.1.

Chaque instruction d'entrée a pour effet d'amener en mémoire centrale l' « image » d'une carte du fichier.

Quand le fichier est sur bande magnétique ou sur disque (1) il doit être découpé en « enregistrements ». L'enregistrement est l'unité d'infor-

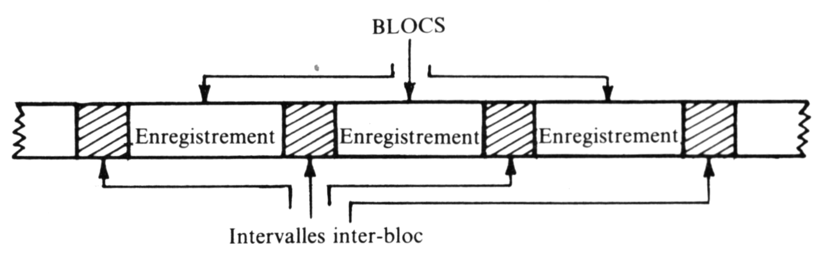


Fig. 10.2.

(1) Au niveau de l'utilisateur les différences de structure interne des enregistrements sur bande ou sur disque n'apparaissent pas.

mation transférée à chaque opération d'entrée ou de sortie : il peut contenir un ou plusieurs caractères, et en particulier un article (cf. paragraphe précédent).

La nature des mémoires auxiliaires accessibles au programme dépend étroitement du mode d'exploitation de l'ordinateur.

Dans une exploitation en centre de calcul la totalité des périphériques peut être disponible.

En temps partagé, outre les instructions d'entrée-sortie pour le terminal, le programmeur dispose très souvent des instructions de transfert sur disque (1).

10.1.3. Mémoires adressables-mémoires non adressables.

La structure technologique des périphériques détermine dans une large mesure la façon de les exploiter.

Avec un lecteur de cartes par exemple, on peut lire une carte seulement après lecture de la carte qui la précède. Un fichier sur cartes doit donc être exploré séquentiellement : une telle mémoire est dite « non adressable ».

Par contre, il est possible de commander au mécanisme d'une unité de disques les opérations suivantes :

— déplacement du bras porte-têtes,

— sélection d'une des têtes.

A la suite de ces opérations, la lecture de la circonférence qui défile sous la tête sélectionnée peut s'effectuer. Cette circonférence appelée « piste » est donc définie par un numéro qui se compose de deux nombres :

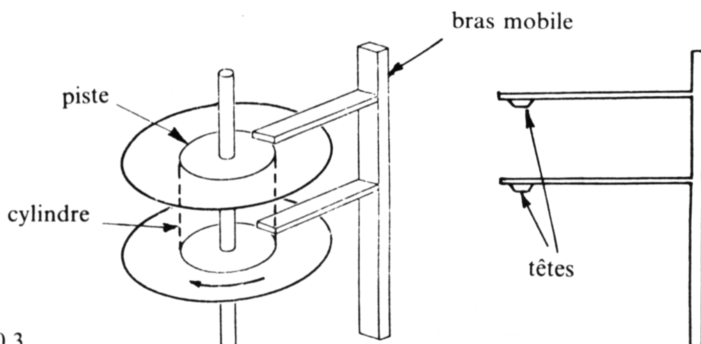


FIG. 10.3.

(1) A titre d'exemple, le système Pdp 11 accepte les instructions de transfert pour :

- unités de disques,
- unités de bandes magnétiques,
- lecteur et perforateur de ruban,
- imprimante,
- lecteur de cartes perforées.

— le numéro de « cylindre » (pour indiquer la position du bras),
— le numéro de tête.

La piste est l'unité de mémoire physiquement « adressable » d'un disque magnétique. Les mémoires se divisent donc en deux grandes catégories, selon qu'elles sont ou non adressables. Corrélativement elles donnent à l'utilisateur des possibilités plus ou moins variées pour accéder à un article.

10.1.4. Adressage d'un enregistrement en BASIC

Le langage BASIC fournit un moyen simple pour donner directement accès à un enregistrement dans une piste (des précisions sur cette méthode d'organisation sont données en 10.2.2).

L'utilisateur peut considérer que chaque enregistrement du fichier porte un numéro qui fait partie de la suite des nombres entiers positifs : 1, 2, 3, ..., *n*.

Dans l'instruction d'entrée ou de sortie, il indique le numéro de l'enregistrement qu'il demande.

A partir de ce numéro, le compilateur et le système d'exploitation calculent l'emplacement physique de l'enregistrement, et transfèrent les données avec la zone de mémoire définie dans le programme de l'utilisateur.

Cette méthode d'adressage est très simple et donne en général de bonnes performances. Elle suppose aussi qu'il existe une relation d'égalité entre l'indicatif (tel qu'il a été défini en 10.1.1) et le numéro d'enregistrement, ce qui est le cas dans bon nombre d'applications.

Quand ceci n'est pas réalisé (séquence d'indicatifs comportant des « trous » ou indicatifs comportant des lettres et des chiffres par exemple), il faut convertir l'indicatif de l'utilisateur (en constituant des tables d'indicatifs par exemple). Le langage COBOL permet de réaliser facilement cette conversion.

10.2. Méthodes d'organisation des fichiers en Basic.

Il existe plusieurs façons de ranger un fichier. Les compilateurs BASIC les plus complets reconnaissent deux méthodes d'organisation :
— l'organisation séquentielle,
— l'organisation directe.

10.2.1. L'organisation séquentielle.

A la création du fichier, les articles sont rangés les uns à la suite des autres sur le support.

L'ordre de classement des articles résulte donc directement de l'ordre suivant lequel la suite des instructions de sortie sont exécutées. En général mais ceci n'est pas indispensable, le programme de création de fichier est conçu pour émettre les commandes de sortie dans l'ordre des indicatifs croissants ou décroissants. Dans ce cas le fichier est dit ordonné : la séquence des indicatifs correspond à l'ordre physique de rangement des articles sur le fichier.

Les traitements ultérieurs du fichier doivent être prévus de façon à accepter les articles dans l'ordre où ils se présentent, sous peine d'aboutir à des durées d'exécution prohibitives (1).

10.2.2. L'organisation directe.

L'organisation directe (2) s'applique uniquement aux mémoires adressables. Dans ce mode, les enregistrements sont rangés sur le support dans l'ordre de leurs numéros.

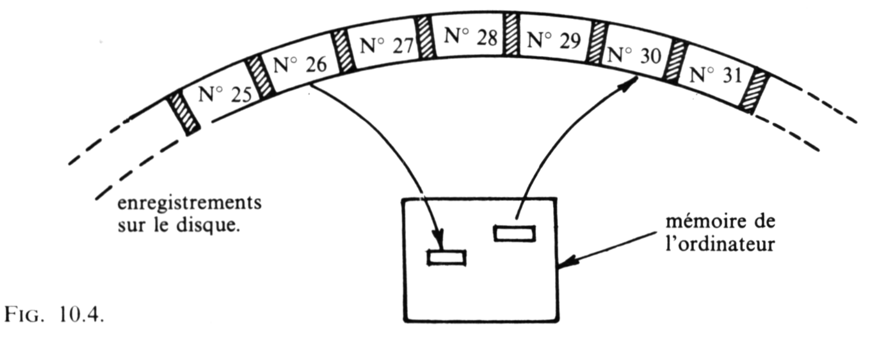


FIG. 10.4.

Quand le programmeur veut effectuer un transfert avec un enregistrement, il indique dans l'instruction le numéro de l'enregistrement désiré. Par exemple :

```
25 INPUT FROM 2 AT 26...
30 PRINT ON 2 AT 30...
```

A la différence de l'organisation séquentielle, le programme de création de fichier peut donc enregistrer les articles dans un ordre quelconque, chaque article allant remplir le bloc correspondant à son numéro.

(1) On peut aussi « trier » le fichier avant le traitement : cette procédure est très couramment mise en œuvre avec les ordinateurs de gestion ; elle nécessite un programme utilitaire spécialisé.

(2) Souvent appelée organisation « relative » pour bien la distinguer de l'organisation « directe avec clés » dans laquelle le système d'exploitation convertit automatiquement les indicatifs en numéros d'enregistrement (langage COBOL par exemple).

En particulier, des blocs peuvent être laissés vacants, pour une utilisation ultérieure.

De même, la consultation du fichier peut s'effectuer sur des numéros d'enregistrement aléatoires.

Enfin, il est possible de mettre à jour un article sans recopier la totalité du fichier.

10.3. Forme des données sur le support.

Les formes les plus courantes d'enregistrement des données sur le support sont les suivantes :

- forme « ASCII » (également appelée « symbolique » ou « télétype »),

et

- forme « binaire » (1).

En forme ASCII, les données sont inscrites en employant un codage du caractère identique à celui du terminal.

En forme binaire, les données sont inscrites en employant un code spécifique à l'ordinateur (2).

Le choix de l'une ou l'autre forme, quand il est laissé à l'utilisateur, entraîne en particulier les conséquences suivantes :

— la forme ASCII permet de créer un fichier puis de l'éditer sous le contrôle d'un programme utilitaire standard disponible dans la plupart des systèmes d'exploitation (et appelé par des commandes indépendantes du langage).

L'emploi de ce type de programme évite un travail de programmation, et apporte aussi diverses aides (possibilités de correction, d'insertion, d'édition partielle).

— les données numériques occupent sur le support une place plus grande quand elles sont exprimées en ASCII que lorsqu'elles sont en binaire. Cette dernière forme est donc plus économique.

(1) La forme EBCDIC, également, est utilisée par IBM.

(2) La plupart des systèmes d'exploitation imposent une limite maximum à la longueur des enregistrements.

Souvent une valeur numérique est représentée sur deux « mots » d'ordinateur.

Pour une chaîne de caractères, une formule (différente pour chaque constructeur) doit être appliquée.

Ainsi pour Hewlett-Packard, le nombre de mots nécessaires pour ranger une chaîne (ayant un nombre impair de caractères) est :

$$1 + ((\text{nombre de caractères de la chaîne}) + 1)/2.$$

— quand les données sont fournies en ASCII puis enregistrées en binaire, l'ordinateur doit effectuer la transformation de code (également pour la présentation des données au terminal, après traitement). Cela peut amener à consommer du temps d'unité centrale.

— certaines méthodes d'organisation de fichiers peuvent être incompatibles avec l'une ou l'autre forme.

Le tableau qui suit présente les différentes combinaisons possibles pour la forme des données et les méthodes d'organisation.

Forme / *Organisation*	ASCII	*Binaire*
Séquentielle	Disponible sur la plupart des compilateurs. Facile d'emploi.	Disponible sur la plupart des compilateurs. Facile d'emploi.
Directe	Disponible sur certains compilateurs (Cegos-Tymshare par exemple). Facile d'emploi quand des programmes utilitaires bien conçus sont disponibles.	Disponible sur la plupart des compilateurs. Assez facile d'emploi.

10.4. Fonctions disponibles pour l'utilisateur.

Un programme utilisant des fichiers comporte en général trois phases principales, comme l'indique l'organigramme qui suit.

A chacune de ces phases correspondent des actions qui intéressent plus particulièrement :

— le système d'exploitation : phase de préparation et phase terminale ;

— le programme utilisateur : phase de traîtement.

Dans une très large mesure, ces actions se retrouvent, identiques, pour les différents constructeurs. Aussi, ce paragraphe décrit en détail les fonctions mises en jeu par l'utilisateur. Divers exemples seront donnés pour montrer la forme générale des instructions.

Au paragraphe 10.5 seront reprises quelques formes particulières pour en préciser l'intérêt dans certains cas précis.

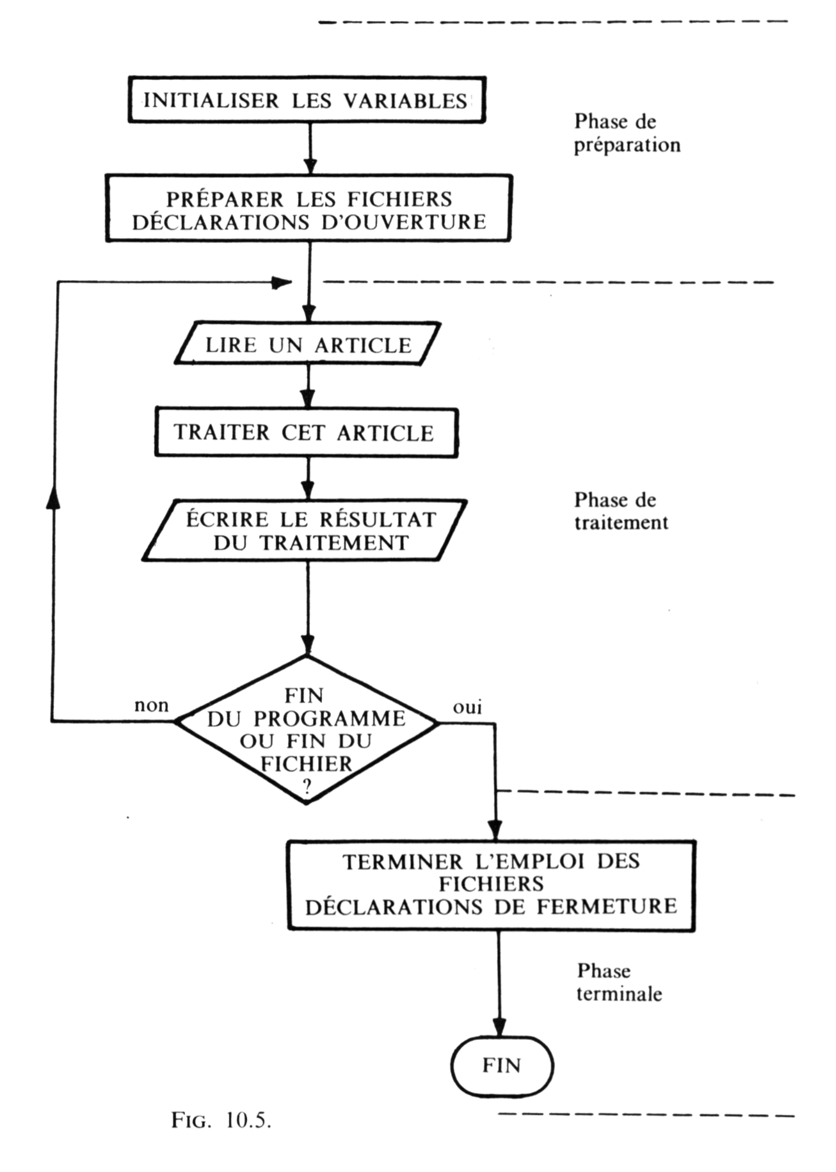


FIG. 10.5.

10.4.1. Phase de préparation : ouverture des fichiers.

Avant toute instruction de lecture ou d'écriture sur un fichier, l'utilisateur doit placer une déclaration d'ouverture qui remplit deux fonctions principales :

— informer le système d'exploitation qu'un fichier portant un certain nom va être utilisé : le système tient à jour un « catalogue » rassemblant

tous les noms et s'assure de l'existence du fichier (quand le fichier est « nouveau », il ajoute simplement le nouveau nom dans le catalogue),

— associer à ce nom une valeur numérique entière qui servira dans la suite du programme à désigner le fichier sous une forme abrégée.

Deux formes pour cette déclaration se rencontrent principalement.

La forme « OPEN » :

Exemple : 10 ØPEN " GAMMA " AS FILE *n*

n est une constante numérique, une variable ou une expression (tronquées si elles ne sont pas entières) qui servira à désigner le fichier.

La forme « FILES » :

Exemple : 10 FILES ALPHA ; BETA ; GAMMA

Les fichiers sont numérotés selon leur ordre de déclaration : ici GAMMA portera le numéro 3.

La forme OPEN permet d'ouvrir un seul fichier par ligne de programme ; elle permet d'apporter des renseignements supplémentaires, en particulier :

— l'usage envisagé pour le fichier (entrée ou sortie ou les deux), par exemple :

10 ØPEN " GAMMA " FOR INPUT AS FILE 3 ;

— la nature du support du fichier, dans les cas où ce choix est laissé à l'utilisateur (dans le cas contraire, le fichier est systématiquement sur disques) ; avec le Pdp 11 par exemple :

10 ØPEN " DTA4 :GAMMA " FØR INPUT AS FILE 3

prend GAMMA sur la quatrième unité de bandes magnétiques ;

— la méthode d'organisation choisie, par exemple :

10 ØPEN " GAMMA ", RANDØM FOR ØUTPUT AS FILE 3

(le fichier GAMMA doit être en organisation directe) ;

— la forme des données sur le support, par exemple :

10 ØPEN " GAMMA ", FØR SYMBØLIC RANDØM ØUTPUT AS FILE 3

(les données seront rangées sous forme symbolique).

10.4.2. Phase de traitement.

Les instructions qui s'appliquent aux fichiers séquentiels et aux fichiers directs ont des formes et des fonctions différentes.

10.4.2.1. Fichiers séquentiels.

Considérons à titre d'exemple un programme destiné à recopier un fichier F1 sur un fichier F2.

L'organigramme est le suivant :

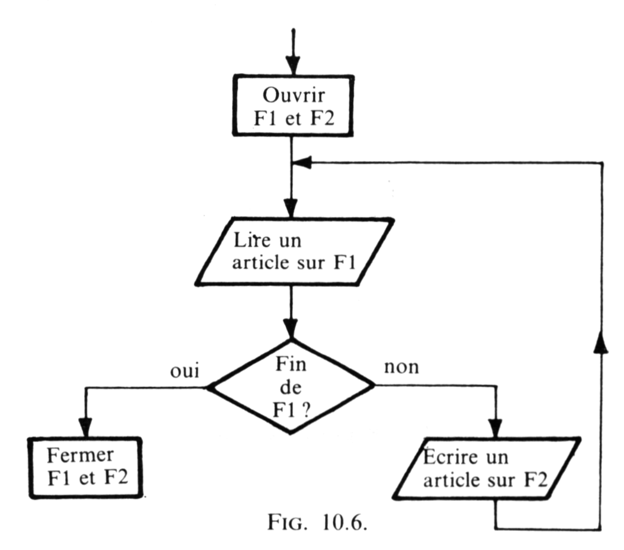


FIG. 10.6.

L'utilisateur doit disposer des fonctions de :

- lecture d'un enregistrement ;
- écriture d'un enregistrement ;
- test de fin de fichier et, éventuellement
- positionnement du fichier au début.

Ce paragraphe décrit les instructions permettant de réaliser ces fonctions.

L'instruction de lecture a pour forme générale :

INPUT ⟨*f*⟩, ⟨ liste ⟩ ou
READ ⟨*f*⟩, ⟨ liste ⟩ (1)

Dans cette instruction :

— *f* (constante numérique ou variable ou expression), tronquée si elle n'est pas entière, donne le numéro du fichier ;

— la liste comprend un ou plusieurs noms de variables (numériques ou caractères, mélangées ou non) éventuellement indicées séparés par des virgules.

(1) Outre des différences mineures de syntaxe, les constructeurs n'admettent pas tous les deux formes INPUT et READ. Quand les deux formes sont disponibles, leurs fonctions sont différenciées (cf. 10.5).

Exemple : 10 FILES ALPHA ; BETA ; GAMMA
20 READ # 3, A, B$, C (1)

Dans cet exemple, trois enregistrements sont lus sur le fichier GAMMA et affectés séquentiellement à A, B$ et C. Les enregistrements trouvés sur le fichier et les variables doivent naturellement être de même nature (numériques ou caractères); dans le cas contraire un message d'erreur est produit.

Si une autre instruction de lecture fait suite à la précédente dans le programme, par exemple :

30 READ # 3, D, E$

la valeur des variables D et E$ est lue à la suite de la valeur de C.

L'instruction d'écriture a pour forme générale :

PRINT ⟨*f*⟩, ⟨ liste ⟩ ou
WRITE ⟨*f*⟩, ⟨ liste ⟩ (2)

Sa syntaxe est la même que pour l'instruction de lecture.

Exemple : 10 FILES ALPHA ; BETA ; GAMMA
20 WRITE # 3, A, B$, C$, D (3)

Dans cet exemple, la valeur des variables A, B$, C$ et D est inscrite sur le fichier GAMMA.

Si une autre instruction d'écriture fait suite à la précédente dans le programme, par exemple :

30 WRITE # 3, E, F$

la valeur des variables E et F$ est écrite à la suite de la valeur de D.

La détection de la fin du fichier se réalise par une instruction dont la forme générale est la suivante :

ØN ENDFILE ⟨*f*⟩ GØ TØ ⟨ numéro de ligne ⟩ ou
IF END ⟨*f*⟩ GØ TØ ⟨ numéro de ligne ⟩ (4)

(1) Avec MARK II la forme des données sur le support est *contrôlée* par la première instruction d'entrée rencontrée : caractère # pour la forme ASCII et caractère : pour la forme binaire.

(2) Quand les deux formes PRINT et WRITE sont disponibles, leurs fonctions sont différenciées (cf. 10.5).

(3) Avec MARK II la forme des données sur le support est *spécifiée* par la première instruction de sortie rencontrée.

(4) Selon les compilateurs, la détection a lieu immédiatement avant ou après l'instruction d'entrée ou de sortie. Le branchement doit donc être placé soit en tête soit à la fin de la boucle de programme.

Cette instruction utilise le principe suivant :

- à la création d'un fichier, le système d'exploitation matérialise automatiquement la fin des données par une marque de fin de fichier ;
- dans un traitement ultérieur (en lecture ou en écriture), l'instruction détecte cette marque, permettant ainsi de sortir de la boucle de programme par un branchement conditionnel.

Positionnement du fichier au début

Dans certains cas, le programme doit pouvoir réexploiter un fichier au début, alors que les traitements précédents l'ont fait progresser de plusieurs enregistrements.

Deux solutions sont couramment proposées :

- fermer le fichier (cf. 10.4.3) et l'ouvrir à nouveau immédiatement après,
- utiliser une instruction spécifiquement définie pour cette fonction (1).

10.4.2.2. Fichiers en organisation directe.

Considérons par exemple un programme permettant de consulter un fichier en organisation directe (F1).

Un numéro est introduit par le terminal, puis :

- l'article qui correspond à ce numéro est lu et
- écrit sur le terminal.

L'organigramme est le suivant (fig. 10.7) :

Pour réaliser cette application (incluant éventuellement une « mise à jour » du fichier), l'utilisateur doit pouvoir :

- lire et écrire un enregistrement correspondant à un numéro donné ;
- positionner un fichier en tout endroit, et
- tester la fin d'un fichier.

Ce paragraphe décrit les instructions qui permettent de réaliser ces fonctions.

L'instruction de lecture a pour forme générale :

INPUT FROM ⟨f⟩ AT ⟨ n ⟩, ⟨ liste ⟩ ou
READ ⟨f⟩, ⟨ n ⟩; ⟨ liste ⟩ (1)

(1) Par exemple : RESTORE # 3 (forme MARK II)
READ # 3,1 (forme HEWLETT-PACKARD)
Des instructions pour positionner un fichier à la fin sont également souvent disponibles : ainsi APPEND (forme MARK II).

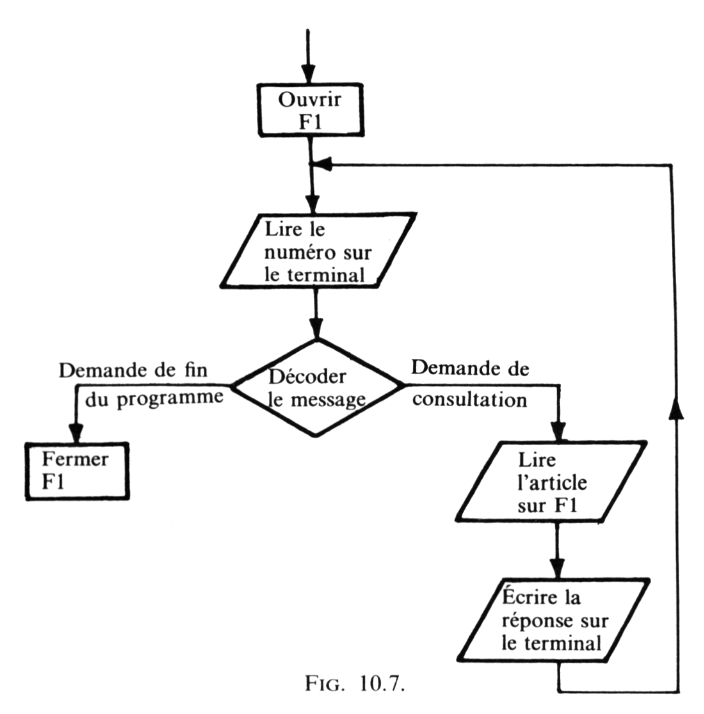


FIG. 10.7.

Dans cette instruction :

- *f* et *n* sont des constantes numériques ou des variables ou des expressions (éventuellement tronquées si elles ne sont pas entières);
- *f* donne le numéro du fichier;
- *n* donne le numéro de l'enregistrement;
- la liste comprend un ou plusieurs noms de variables (numériques ou caractères, mélangées ou non) éventuellement indicées séparées par des virgules.

Exemple : 20 READ # 3, 54; A, B$, C

Dans cet exemple, l'enregistrement de numéro 54 est lu et les valeurs qu'il contient sont affectées à A, B$ et C.

Comme pour les fichiers séquentiels, données et variables doivent être de même nature.

L'instruction d'écriture a pour forme générale :

OUTPUT TO ⟨*f*⟩ AT ⟨*n*⟩, ⟨liste⟩ ou
PRINT ⟨*f*⟩, ⟨*n*⟩; ⟨liste⟩ (1)

Sa syntaxe est la même que pour l'instruction de lecture.

(1) La forme des instructions varie selon les compilateurs mais la signification de *f* et *n* demeure inchangée.

Progression automatique du numéro d'enregistrement :

Après une instruction de lecture ou d'écriture, le numéro d'enregistrement progresse automatiquement de 1.

Cela permet de réaliser un traitement séquentiel sur un fichier direct.

Exemple :

```
..........................................
..........................................
30 N = 1
40 FØR I = 1 TØ 40
50 PRINT #3, N ; A$
..........................................
90 NEXT I
..........................................
```

Il est également possible de faire progresser le numéro d'enregistrement au moyen d'une instruction FOR.

Détection de la fin de fichier :

Quand un traitement séquentiel est appliqué à un fichier en organisation directe, les instructions de détection de fin de fichier peuvent être utilisées.

Positionnement sur un enregistrement donné et reconnaissance du numéro d'enregistrement :

En plus des instructions précédentes, la plupart des compilateurs offrent des instructions permettant de :

- positionner un fichier sur un enregistrement donné ; ainsi :

LØCATE 3 ØN 4 (1)

positionne le fichier numéro 4 sur le troisième enregistrement ;

- connaître la position sur laquelle le fichier se trouve actuellement, ainsi :

```
....................................... (1)
LOCATE 3 ON 4
PRINT " LA POSITION ACTUELLE EST : ", LOC (4)
.......................................
```

donne pour résultat à l'exécution :

```
LA POSITION ACTUELLE EST : 4
```

(1) Formes XDS-940.

10.4.2.3. Fichiers en organisation directe considérés comme extension de mémoire.

Les programmes de calcul scientifiques produisent souvent des résultats intermédiaires en quantités telles que leur stockage en mémoire centrale n'est pas économiquement possible. Il est alors intéressant de les ranger sur un fichier externe en organisation directe, pour obtenir une extension de la capacité « apparente » de la mémoire.

Avec des fichiers constitués de cette manière, le BASIC considère que les articles sont les éléments de tableaux, et emploie certaines instructions qui leur étaient réservées.

Dans l'exemple suivant :

```
10 ØPEN « DØNNÉES » AS FILE 1
20 DIM #1, A(100, 20), B(1000), C$(500)        (1)
```

le fichier DONNÉES est constitué par :

- le tableau A (100 lignes et 20 colonnes),
- la liste B (1 000 éléments),
- 500 chaînes de caractères.

Dans la suite du programme, l'accès à un élément particulier s'opère par les instructions habituelles.

Par exemple :

```
50 INPUT #1, A(28, 13)
```

Les instructions MAT INPUT et MAT PRINT peuvent également être utilisées. On peut écrire par exemple :

```
50 MAT INPUT #1, A(28, 13)
```

10.4.3. Phase terminale. Fermeture des fichiers.

Quand un programme cesse d'utiliser un fichier, il doit exécuter une instruction de fermeture pour :

- informer le système d'exploitation qu'un fichier est libéré, et
- lui demander de repositionner ce fichier au début.

La forme générale de cette instruction est la suivante :

CLØSE ⟨ *f*1 ⟩, ⟨ *f*2 ⟩

dans laquelle *f*1 et *f*2 sont les numéros des fichiers fermés.

Les systèmes d'exploitation imposent tous une limite au nombre de fichiers ouverts simultanément (souvent 4). L'instruction CLOSE peut permettre à un programme d'utiliser un nombre supérieur de fichiers (à la condition que la succession de fermetures et de ré-ouvertures ne nuise pas trop aux durées d'exécution).

(1) Forme Pdp 11.

10.5. Quelques possibilités de compilateurs Basic évolués.

Certains compilateurs (1) offrent des possibilités intéressantes pour le traitement des fichiers séquentiels ou directs.

Les principales sont regroupées dans ce chapitre ; sur des exemples, des détails complémentaires concernant des formes syntaxiques diverses seront précisées.

10.5.1. Fichiers séquentiels.

Ce paragraphe rassemble les fonctions relatives aux fichiers séquentiels.

10.5.1.1. Création des fichiers et instructions d'entrée.

Considérons le fichier ASCII de nom ENT, dont le contenu est le suivant :

```
10 1,ABC,2,DEF,3,GHI ⮐
20 4,JKL,5,MNØ,6,PQR ⮐
```

Comme on l'a vu en 10.2, ce fichier a pu être créé en utilisant un programme utilitaire standard. Les articles 1 ABC 2, etc., ont été introduits directement à partir du terminal. Ils doivent être séparés, et pour cela on emploie le délimiteur standard : une virgule. A la fin d'une ligne le caractère « Retour-Chariot » (2) joue le rôle de délimiteur d'article et en plus de délimiteur de ligne comme le montre l'exemple ci-après.

En outre, les lignes sont numérotées, comme pour un programme.

Ce programme réalise des entrées à partir du fichier ENT et affecte des valeurs aux variables A, B$, C et D$ (fig. 10.8).

La commande FILES de la ligne 10 est la déclaration d'ouverture (forme particulière de ØPEN pour MARK II) de ENT, et sa désignation interne est 1.

Le signe # spécifie que le format est ASCII (il s'introduit ici au niveau des instructions d'entrée-sortie ; pour le format binaire ce signe est :).

Le résultat des deux instructions INPUT est donné dans le tableau : les numéros de ligne sont pris comme partie intégrante des articles. De plus, INPUT distingue le séparateur virgule du séparateur ⮐ : chaque instruction affecte un début de ligne à la première variable de la liste.

(1) MARK II (Honeywell-Bull) et SUPERBASIC (CIGI-Tymshare) ont été pris en exemple.

(2) Ce caractère correspond à la touche « CR » du terminal : dans la suite de ce chapitre, il est représenté par le symbole ⮐ sur les feuilles produites par le terminal, quand cela est indispensable pour la compréhension du texte.

READ, par contre ne prend pas en compte le numéro de ligne, et ne distingue pas les deux séparateurs.

On note enfin l'instruction RESTØRE qui repositionne le fichier au début, dans le cours du programme.

```
10 FILES ENT
20 INPUT #1,A,B$,C
30 INPUT #1,A,B$,C
40 RESTØRE #1
50 READ #1,A,B$,C,D$
60 READ #1,A,B$,C,D$
```

Par ligne nº	*Valeurs affectées aux variables* A	B$	C	D$
20	101	ABC	2	—
30	204	JKL	5	—
50	1	ABC	2	DEF
60	3	GHI	4	JKL

FIG. 10.8.

10.5.1.2. Instructions de sortie.

Utilisons encore le fichier ENT, et ouvrons en plus les fichiers SØR et ØUT :

```
10 FILES ENT,SØR,ØUT
20 SCRATCH #2
30 SCRATCH #3
40 READ #1,A,B$,C
50 WRITE #2,A,B$,C
60 PRINT #3,A,B$,C
```

La déclaration d'ouverture affecte respectivement à ENT SØR et ØUT les désignations internes 1, 2 et 3.

De même qu'en entrée, les instructions WRITE et PRINT ont des effets différents, comme le montre le résultat d'édition de SØR et ØUT, ci-dessous :

```
SØR
100 101,              ABC,              2,

ØUT
101              ABC              2
```

Au niveau de WRITE et de PRINT les éléments de la liste peuvent être séparés par une virgule ou un point-virgule (les effets de ces séparateurs sont décrits au paragraphe 5.4.1).

On remarque enfin les deux instructions SCRATCH qui sont rendues nécessaires par le fait que FILES ouvre tous les fichiers en entrée seulement. Elle évite la destruction accidentelle d'un fichier en obligeant l'utilisateur à rappeler qu'il désire écrire sur un de ses fichiers.

10.5.1.3. Instructions de sortie avec image.

Comme pour PRINT (voir 5.5) il est possible de réaliser la mise en page par une instruction IMAGE.

Par exemple :

```
10: ##.## ##.## ##.##
20 FILES IMAG1
30 SCRATCH #1
..........
..........
80 PRINT #1 USING 10,A(1),A(2),A(3)
```

10.5.2. Fichiers ASCII en organisation directe.

Ce paragraphe décrit en détail certaines instructions relatives à des fichiers ASCII en organisation directe (1). La plus petite partie d'un fichier, ou élément est le caractère.

Groupés, plusieurs éléments constituent un article. La longueur d'un article est le nombre de caractères qu'il contient.

10.5.2.1. Articles de longueur variable ou fixe.

Le numéro d'enregistrement a une signification quelque peu différente suivant que le fichier est organisé en articles de longueur variable ou de longueur fixe.

Articles de longueur variable :

Chaque *élément* est considéré comme un enregistrement, et, comme tel, porte un numéro.

L'exemple qui suit représente un fichier ASCII et indique les numéros de certains éléments.

CE FICHIER EST ↵

UN EXEMPLE : 1↵

2 3 4 5↵

Elément	*Numéro*
F	4
X	20
3	32

FIG. 10.9.

Il convient de remarquer que le caractère ↵ est compté comme élément.

Articles de longueur fixe :

Le plus souvent, l'utilisateur désire retrouver aisément dans un fichier non pas un élément, mais un *article*.

(1) XDS-940. CIGI-Tymshare.

Quand le langage lui donne la possibilité d'employer des fichiers avec des articles de longueur fixe, la déclaration d'ouverture comporte la mention de la longueur de l'enregistrement. Dans l'exemple qui suit :

ØPEN « STØCK », RANDØM (26) IØ, 1

les articles du fichier STOCK ont une longueur constante de vingt-six caractères.

Dans ce cas, la notion d'élément n'a plus d'utilité en ce qui concerne l'adressage de la mémoire auxiliaire, puisque l'utilisateur accède directement à la totalité de l'article. Pour le traitement, elle permet néanmoins de séparer l'enregistrement en parties (sous-articles).

10.5.2.2. Formats d'entrée et de sortie.

Les instructions d'entrée-sortie pour les fichiers peuvent être assorties de formats (1) qui apportent une très grande souplesse pour *individualiser* une partie d'un article et également pour *controler sa nature* (alphabétique ou numérique par exemple).

Considérons par exemple le fichier ASCII :

PIERRE, 3.25
JEAN, 2.75
PHILIPPE, 4.90
XAVIER, 2.50

On peut obtenir deux articles complets par le programme suivant :

```
10 ØPEN " NØMS ", RANDØM INPUT, 2
20 INPUT FRØM 2 AT 14 IN FØRM " R " A, B.
```

L'instruction 20 affecte à A et B les valeurs suivantes :

A... JEAN, 2.75
B... PHILIPPE, 4.90

Un autre format permet d'obtenir un élément alphabétique ou numérique :

```
20 INPUT FROM 2 AT 14 IN FØRM "X" :A
```

affecte à A la valeur J.

Un autre format permet de traiter la partie numérique d'un article :

```
10 ØPEN " NØMS ", RANDØM IØ, 3
20 PRINT ØN 3 AT 19 IN FØRM "BD.DD" :2.85
```

et le deuxième article est devenu :

JEAN, 2.85

(1) La signification des descripteurs les plus intéressants est précisée au paragraphe 11.10.4.

10.6. Exemple d'application : un inventaire permanent.

On désire tenir l'inventaire permanent d'un magasin d'outillage et de pièces détachées diverses.

Le traitement doit permettre de faire la mise à jour quotidienne du fichier « STOCK » en éditant un état concernant seulement les articles mouvementés.

10.6.1. Description des fichiers et principe du traitement.

Deux fichiers seront ouverts :

- Le fichier « MOUVEMENT » :
  - organisation : séquentielle, non ordonné sur les numéros d'articles;
  - forme des données : ASCII;
  - enregistrements : numéro d'article suivi de la quantité mouvementée (positive ou négative).
- Le fichier « STOCK » :
  - organisation : directe; le numéro de l'article est le numéro de l'enregistrement;
  - forme des données : ASCII.
  - enregistrements : longueur fixe (26 caractères) comprenant :
  - le libellé : 21 caractères;
  - la quantité en stock : 4 caractères;
  - un caractère « retour-chariot ».

La lecture d'un enregistrement sur « MOUVEMENT » donne un numéro d'article qui permet d'appeler l'enregistrement correspondant de « STOCK » et de le mettre à jour; une ligne de l'état est éditée.

Ce traitement se poursuit jusqu'à la fin du fichier « MOUVEMENT ».

10.6.2. Organigramme (fig. 10.10).

10.6.3. Remarques concernant la programmation.

Des détails concernant la forme des instructions de ce programme sont donnés au paragraphe 11.10.3.

On notera de plus les points suivants :

- l'édition de « STOCK » fait apparaître des 0 pour les articles mouvementés car ce programme a été exécuté plusieurs fois et comporte un descripteur 4D dans l'instruction de mise à jour;

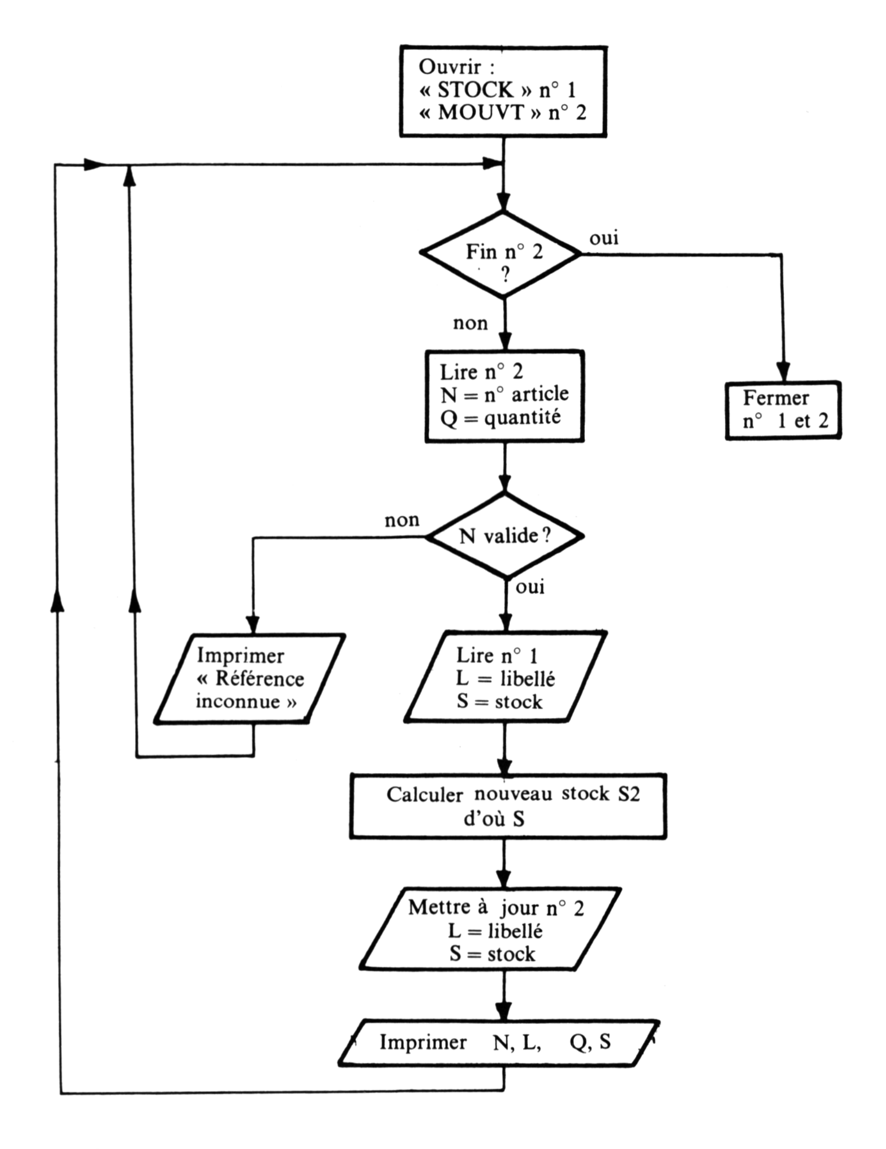


FIG. 10.10.

● l'enregistrement est de longueur 26 pour tenir compte du caractère retour-chariot qui a été introduit à la création du fichier (un programme utilitaire standard a été employé) ;

● les fonctions VAL et STR ont été employées pour convertir les variables caractères en variables numériques et *vice-versa*.

10.6.4. Édition des fichiers. Programme et résultat d'exécution.

Ficher « MOUVEMENT » (édition) :

```
7,-8,4,-2,10,-3,9,5,12,20,2,200
```

Fichier « STOCK » (édition avant exécution du programme) :

```
VIS TF ACIER 4/40       600
VIS TF ACIER 4/45      0310
ECROU ACIER 4           800
CLE PLATE 8/10         0166
CLE PLATE 10/12         020
TOURNEVIS 3/120         030
TOURNEVIS 8/120        0164
PINCE UNIVERSELLE 160 015
PINCE COUPANTE 120     0235
FORET ACIER RAPIDE 4 0349
FORET ACIER RAPIDE 5   150
FORET ACIER RAPIDE 6 0640
```

Programme :

```
0   STRING S
10  PRINT "REFERENCE              LIBELLE            MOUVEMENT     STOCK RESTANT"
20  PRINT
30  OPEN "STOCK",RANDOM (26) IO,1
40  OPEN "MOUVEMENT",INPUT,2
50  ON END FILE (2) GO TO 130
60  INPUT FROM 2:N,Q
65  IF N>12 THEN 150
70  INPUT FROM 1 AT N IN FORM '21X 4D':L,S
80  S2=VAL(S)+Q
90  S=STR(S2)
100 PRINT ON 1 AT N IN FORM '21X 4D':L,S
110 PRINT IN FORM "3B 2% 10B 21% 8B 4% 13B 4%/":N,L,Q,S
120 GO TO 50
130 CLOSE 1,2
140 END
150 PRINT IN FORM "3B2%5B25X/":N,"****REFERENCE INCONNUE****"
160 GO TO 50
```

Résultat de l'exécution :

REFERENCE	LIBELLE	MOUVEMENT	STOCK RESTANT
7	TOURNEVIS 8/120	-8	156
4	CLE PLATE 8/10	-2	164
10	FORET ACIER RAPIDE 4	-3	346
9	PINCE COUPANTE 120	5	240
12	FORET ACIER RAPIDE 6	20	660
2	VIS TF ACIER 4/45	200	510

10.6.5. Remarques concernant l'application.

L'application a été simplifiée à l'extrême pour alléger l'exposé.

A titre d'exercice on pourra par exemple envisager :

- des contrôles de validité plus nombreux,
- l'introduction du numéro d'article à l'intérieur des enregistrements de « STOCK » pour le comparer avec le numéro d'enregistrement,
- la préparation d'un fichier « historique » pouvant par la suite servir à établir des statistiques d'entrée et de sortie de matériels,
- un codage « significatif » du numéro d'article pour le convertir en numéro d'enregistrement (exercice difficile).

CHAPITRE 11

LES PRINCIPALES EXTENSIONS DU BASIC

11.0. Introduction.

Dès qu'un nouveau langage de programmation est mis à la disposition des utilisateurs, ceux-ci l'emploient pour des applications d'abord relativement simples puis de plus en plus compliquées. Des lacunes sont alors mise en évidence. Poussés par une concurrence sévère, les constructeurs les comblent progressivement.

Cette évolution s'est produite surtout pour les principaux langages de programmation, ainsi :

FORTRAN : Fortran II, puis Fortran IV, ensuite Fortran 1977 ; des extensions sont déjà envisagées pour 1981/82.

ALGOL : ALGOL 58, puis ALGOL 60, ALGOL-W et enfin ALGOL 68.

Pour BASIC, cette évolution ne s'est pas traduit par de nouvelles normes (la norme a été proposée tardivement). En revanche de nombreuses extensions ont été apportées ; le tableau suivant en présente les principales catégories.

Types d'extensions	*Exemples d'extensions*
BASIC « originel »	*Chaînes de caractères* *instructions « matricielles »*
extensions « Algorithmiques »	IF THEN ELSE FOR WHILE
extensions « Industrielles »	commande du Bus GP-IB
extensions « Graphiques »	tracé de courbe
extensions « Gestions »	{ fichiers séquentiels { fichiers à accès direct
extensions « Home Computer » (ordinateur individuel)	commande du téléviseur

Comme il n'est pas possible de faire une liste exhaustive de toutes les extensions rencontrées sur l'ensemble des systèmes commercialisés. Ce chapitre se limite aux extensions les plus significatives.

11.1. Les constantes et les variables.

Le BASIC classique est conçu pour accepter deux types de variables et de constantes :

— variables et constantes numériques,

— variables et constantes « caractères ».

Les variables numériques sont toutes représentées en virgule flottante, ce qui simplifie la réalisation du software (compilateur, interpréteur, etc.) au détriment de l'encombrement mémoire et de la vitesse d'exécution.

11.1.1. Différents types de variables.

Certains systèmes offrent la possibilité d'utiliser d'autres types de variables :

- des variables entières,
- des variables logiques,
- des variables « double précision ».

Tout ceci a nécessité des extensions et des modifications au BASIC original.

*Système Pdp*11 *:* Les variables entières (représentées en virgule fixe sur 16 bits au lieu de 32) sont caractérisées par un identificateur suivie du caractère %.

Exemple : A %, B1 % sont des variables entières. L'utilisation de ces variables (lorsque cela est possible) permet de diminuer l'encombrement mémoire et d'améliorer les performances à l'exécution.

Systèmes CDC et XDS 940 *:* A partir d'instructions de déclaration INTEGER, LOGICAL, COMPLEX, STRING et TEXT on peut spécifier le type des variables et même déclarer des tableaux :

Exemple :

```
0010 LØGICAL L1,L(6),E,F
0020 CØMPLEX S,Y(20)
     .
     .
     .
0100 L(1)= E AND F
0110 L1= A+C*B
     .
     .
```

L'instruction 10 indique que L1 est une variable logique et que L est un tableau comportant six éléments dont l'indice va de 1 à 6.

L'instruction 20 indique que X et Y sont respectivement une variable simple et un tableau de type complexe.

Les variables LOGIQUES sont utilisés essentiellement pour mémoriser le résultat de comparaison et pour effectuer des calculs booléens.

L'encombrement des tableaux LOGIQUES est très faible : un bit par élément.

Les variables logiques sont utilisées à partir d'opérateurs logiques notamment AND NOT et OR (comme en FORTRAN IV). On peut aussi attribuer aux variables logiques simples (pas aux tableaux) des valeurs numériques. Une variable logique est fausse si sa valeur est zéro et elle est vraie si sa valeur numérique est différente de zéro.

Les variables COMPLEXES sont destinées à permettre des calculs en « nombres complexes » comme en mathématiques. Dans ce but elles sont représentées par un couple de deux nombres réels (partie réelle et partie imaginaire). L'utilisation des variables complexes nécessite des fonctions standards complémentaires (CMPLX par exemple). Leur utilisation est exposée au paragraphe 11.12.

Au système XDS 940 ont été adjointes des variables DOUBLE PRÉCISION, qui sont déclarées au moyen de la déclaration DOUBLE. Il s'agit de variables réelles dont la représentation en mémoire plus encombrante permet d'atteindre une meilleure précision. Compte tenu de la longueur du mot mémoire des ordinateurs CDC, cette extension a été jugée inutile.

REMARQUE : Le système XDS 940 de Télésystèmes accepte en outre des variables entières double précision dont l'intérêt est assez restreint.

11.1.2. Nombre d'indices, leur plage de variation.

Le BASIC originel était conçu pour accepter des variables numériques comportant 0,1 ou 2 indices et des variables caractères à 0 ou 1 indice. Certains systèmes acceptent des variables comportant trois indices et plus (XDS 940, CDC 6000 et Cyber par exemple).

Normalement le ou les indices varient de 1 à N, N étant la constante figurant dans la déclaration DIM. Cette méthode inspirée de FORTRAN présente pour certaines applications, quelques difficultés et quelques systèmes offrent des solutions de rechange.

Solution DEC, MARK II et Data General. Les indices vont de 0 à *n*. Cette méthode présente le principal inconvénient d'être incompatible avec les autres systèmes.

Solution XDS 940 *et CONTROL DATA* 6000. L'instruction DIM admet deux formes syntaxiques : la forme habituelle permettant de faire varier les indices de 1 à *n*, une nouvelle forme similaire à ce qui se fait en ALGOL :

DIM A(– 5 : 10, 15 : 20)

signifie que le tableau A a deux indices dont le premier varie de – 5 à 10 et le second de 15 à 20.

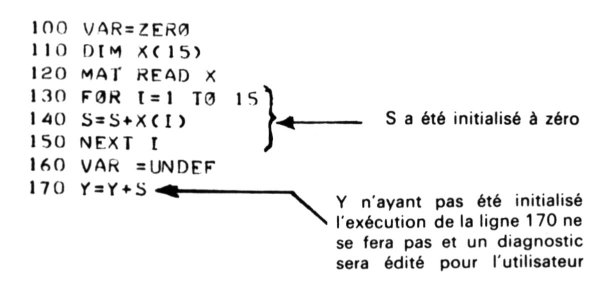
Cette solution est très souple et permet la compatibilité avec les systèmes moins évolués.

11.1.3. Initialisation des variables.

Pour disposer des avantages dus à une initialisation systématique à zéro et d'une non-initialisation (cf. paragraphe 3.2) deux systèmes (XDS 940 et CDC) offrent à l'utilisateur le choix entre ces deux possibilités au moyen des instructions.

VAR = UNDEF et VAR = ZERØ.

Exemples :



```
100 VAR=ZERØ
110 DIM X(15)
120 MAT READ X
130 FØR I=1 TØ 15
140 S=S+X(I)
150 NEXT I
160 VAR =UNDEF
170 Y=Y+S
```


11.2. Présentation d'un programme.

Nous avons vu qu'en « BASIC CLASSIQUE », chaque ligne comporte un numéro de ligne qui tient lieu d'étiquette et une seule instruction.

Cette méthode très simple allonge considérablement la longueur d'un programme. Pour obvier à cet inconvénient, certains systèmes acceptent que l'utilisateur écrive plusieurs instructions sur une même ligne à condition qu'elles soient séparées les unes des autres par un *séparateur* qui est souvent une virgule (XDS 940, CDC 6000, CII 10070) ou parfois deux points (Pdp 11).

Une instruction pourra figurer sur plusieurs lignes consécutives. Dans ce cas chaque ligne nécessitant une ligne suite se termine par le caractère Line Feed au lieu de « Carriage Return ».

En outre, le préfixe LET peut très souvent être omis sur la plupart des systèmes.

Ceci conduit à la présentation de programmes ayant l'allure suivante :

```
090 .....
100 X=A, Y=0, PI=3.14159
110 .....
```

Dans ce cas, seule la première instruction de la ligne est « étiquetée » et on ne peut donc pas effectuer un branchement directement aux autres instructions de cette ligne sans passer par la première.

11.3. Instruction IF généralisée.

La forme initiale de l'instruction IF étudiée aux chapitres 3 et 8 avait l'allure suivante :

IF ⟨ comparaison ⟩ THEN ⟨ numéro de ligne ⟩.

Cette forme rend les programmes peu lisibles et allongent parfois un programme. Afin d'éviter des lourdeurs. des extensions ont été apportées notamment sur les systèmes XDS 940, CDC, Pdp 11.

11.3.1. Première extension :

La forme suivante est acceptée

IF ⟨ comparaison ⟩ THEN ⟨ Instruction exécutable ⟩.

Cette forme permet d'utiliser toutes les instructions exécutables sauf l'instruction FOR.

L'interprétation est la suivante : si le test est satisfait, l'instruction qui suit THEN est exécutée sinon il y a passage à la ligne suivante.

Ceci explique que l'instruction exécutable qui suit THEN ne puisse pas être une instruction FOR. En effet dans le cas où le test ne serait pas satisfait, il y aurait branchement à l'intérieur de la portée d'une boucle sans que l'instruction FOR d'initialisation soit exécutée (voir règles relatives aux boucles de calcul au chapitre 4).

La traduction en BASIC de l'organigramme suivant montre l'intérêt de cette extension :

X = 0
oui
non
Y = 3

FIG. 11.1.

sans extension	avec extension
100 IF X ⟨ ⟩ 0 THEN 120	100 IF X = 0 THEN Y = 3
110 Y = 3	
120	

11.3.2. Deuxième extension.

Cette extension inspirée d'ALGOL 60 permet de rédiger un programme sous un aspect plus élégant. La forme générale de cette instruction est

IF ⟨ comparaison ⟩ THEN ⟨ numéro de ligne ou instruction exécutable ⟩ ELSE ⟨ numéro de ligne ou instruction exécutable ⟩

Exemple 1 : IF X = 5 THEN 200 ELSE 300

si X = 5 il y a branchement à la ligne 200 sinon il y a branchement à la ligne 300.

Exemple 2 : IF X = 5 THEN Y = 3 ELSE C = 2.

Cette instruction traduit l'organigramme ci-contre.

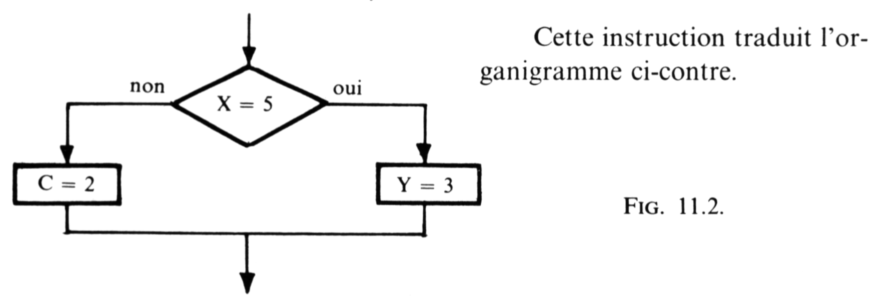


FIG. 11.2.

Puisque IF constitue une instruction exécutable, il est possible de combiner plusieurs instructions IF entre elles afin d'augmenter la concision des programmes :

IF... THEN... ELSE IF... THEN... ELSE...
IF... THEN IF... THEN... ELSE... ELSE...

Exemple 3 : Traduire les organigrammes suivants au moyen d'une seule instruction

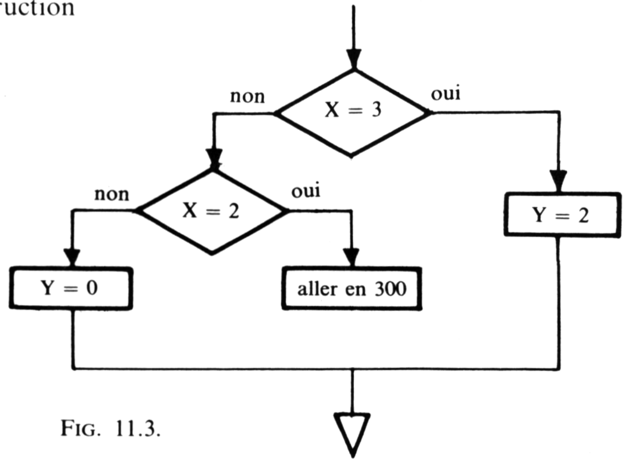


FIG. 11.3.

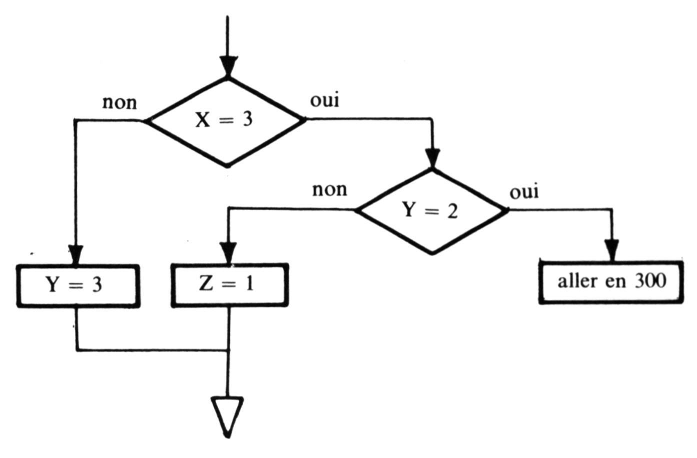


FIG. 11.4.

IF X = 3 THEN Y = 2 ELSE IF X = 2 THEN 300 ELSE Y = 0
IF X = 3 THEN IF Y = 2 THEN 300 ELSE Z = 1 ELSE Y = 3

11.3.3. Troisième extension.

A l'intérieur d'une instruction IF, il est possible d'insérer plusieurs instructions d'affectation consécutives.

Exemple : IF X = 3 THEN Y = 2, Z = 3 ELSE C = T, D = T/N.

11.4. Les fonctions non standards.

Différentes extensions ont été apportées : d'une part l'acceptation de fonctions de plusieurs variables, d'autre part l'acceptation de fonctions se définissant au moyen de plusieurs instructions.

11.4.1. Fonction de plusieurs variables.

Nous pouvons écrire

FNA(X, Y, Z) = ⟨ expression arithmétique ⟩.

Cette extension est acceptée par les systèmes

XDS Sigma 5/7, Pdp 11, XDS 940, CDC.

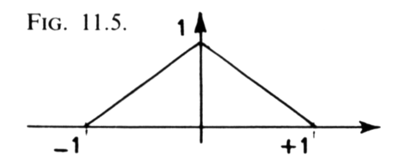
11.4.2. Fonction nécessitant plusieurs instructions.

La séquence d'instructions permettant de définir la fonction sera située entre la déclaration DEF FN... et l'instruction FNEND qui termine

la définition de la fonction. Entre ces deux instructions on pourra écrire une séquence d'instructions quelconque.

Exemple : Pour définir la fonction $f(x)$ dont le graphe est donné ci-contre nous écrivons sous forme mathématique

FIG. 11.5.



$$f(x) = 0 \quad \text{pour } x \leqslant -1 \text{ et } x \geqslant 1$$
$$f(x) = 1 + x \quad \text{pour } -1 \leqslant x \leqslant 0$$
$$f(x) = 1 - x \quad \text{pour } 0 \leqslant x \leqslant 1.$$

En BASIC la définition pourra être la suivante

```
0010 DEF FNF(X)
0020 FNF=0
0030 IF X>-1 AND X<0 THEN FNF=1+X
0040 IF X>0 AND X<1 THEN FNF=1-X
0050 FNEND
```

REMARQUE

1. Les fonctions à une seule variable, seules admises dans le BASIC original étaient d'une utilisation peu commode.

2. Ces deux extensions sont à comparer avec la notion de fonction arithmétiquement définie et les « FUNCTION » du FORTRAN IV.

11.4.3. Exercice.

3. Écrire la séquence précédente de façon plus concise en utilisant des formes étendues du IF :

```
10 DEF FNF(X)
20 IF X<-1 ØR X>1 THEN FNF=0 ELSE IF X<0 THEN FNF=1+X ELSE FNF=1-X
30 FNEND
```

11.5. Instruction FOR avec liste de valeurs.

Lorsqu'il faut répéter une séquence de calcul pour une suite de valeurs d'un paramètre dont la progression n'est pas constante, l'instruction FOR classique n'est pas utilisable. Pour combler cette lacune, certains systèmes (XDS 940...) offre une forme étendue très commode dont la syntaxe est la suivante :

```
FØR N = liste de valeurs
NEXT N
```

ou

```
FØR N = liste de valeurs TØ...
NEXT N
```

N prendra successivement toutes les valeurs de la liste. Un élément de la liste accompagné de la séquence TO est interprété comme un FOR classique :

Exemple :

```
100 FØR N=2,3,8 TØ 14 STEP 2,50,60
......
400 NEXT N
```

La séquence sera répétée pour N prenant les valeurs :

2, 3, 8, 10, 12, 14, 50, 60.

Remarque : Cette instruction est à comparer avec l'instruction REPEAT du FORTRAN XDS sigma 5/7 et 9300.

11.6. Forme dynamique de l'instruction FOR.

La forme initiale de l'instruction FOR est régie notamment par la règle suivante (cf. 4.4) :

FØR I = M1 TØ M2 STEP M3.

Si M2 et M3 sont des expressions, ces expressions sont « évaluées » avant d'entrer dans la portée de la boucle et par conséquent la progression de I se fait avec pas constant et la limite ne peut être modifiée pendant que les instructions situées dans la portée sont exécutées.

Cette forme « statique » permet une compilation plus simple et surtout dans la plupart des cas de meilleures performances à l'exécution.

Toutefois pour certaines applications une forme « dynamique » s'avère plus commode et pour cela certains systèmes (XDS 940-Pdp 11-CDC) offrent deux formes nouvelles qui sont :

FØR... WHILE... et FØR... UNTIL...

11.6.1. Forme FOR ... WHILE.

La forme de cette instruction est la suivante :

PORTÉE de la boucle {
FOR ⟨ var ⟩ = ⟨ exp ⟩ WHILE ⟨ condition ⟩
⋮
NEXT ⟨ Var ⟩.
}

Les instructions de la portée sont exécutées tant que la condition est satisfaite mais comme pour la forme suivante, à chaque itération, les expressions qui peuvent être éventuellement contenues dans la condition sont calculées.

11.6.2. Forme FOR ... UNTIL.

La forme de cette instruction ressemble beaucoup à la précédente.

FOR ⟨ var ⟩ = ⟨ exp ⟩ STEP ⟨ expression ⟩ UNTIL ⟨ condition ⟩
⋮
NEXT ⟨ Var ⟩.

Par contre la signification est différente : la portée est répétée jusqu'à ce que la condition soit satisfaite et alors on continue en séquence.

Ces deux formes peuvent être utilisées également en tant que « modificateurs d'instructions ».

11.7. Next multiple.

Lorsque deux boucles imbriquées se terminent en même temps, le programme contient deux instructions NEXT consécutives :

FOR I = ...
....
FOR J = ...
....
{ NEXT J
{ NEXT I } = NEXT J, I

Certains systèmes (XDS 940, CDC) acceptent que l'utilisateur remplace ces deux instructions par une seule qui contient les deux variables à condition toutefois de respecter leur ordre d'apparition.

11.8. Les modifications d'instructions.

Pour permettre l'obtention de programmes plus concis, l'utilisateur peut ajouter à la plupart des instructions des modificateurs qui précisent dans quelles conditions l'instruction modifiée doit être exécutée.

Aux instructions modifiables, on peut ajouter un nombre quelconque de modificateurs, qui dans ce cas sont évalués dans l'ordre inhabituel de la droite vers la gauche.

Exemple : PRINT " X = "; X IF X > Y UNLESS X < 0.

Si X est négatif, il y a passage immédiat à l'instruction suivante ; sinon le test X > Y est effectué, si la réponse est négative il y a passage à l'instruction suivante. Si la réponse est positive, l'instruction PRINT est exécutée et il y a alors passage à la prochaine instruction.

Les modificateurs disponibles sont IF, UNLESS, UNTIL, WHILE et FOR.

11.8.1. Modificateur IF.

L'instruction suivie du modificateur IF est exécutée seulement si la condition qui suit le IF est satisfaite. Ensuite l'instruction suivante sera exécutée.

11.8.2. Modificateur UNLESS.

L'instruction suivie du modificateur UNLESS est exécutée seulement si la condition n'est pas satisfaite.

REMARQUE : Utilisés seuls, ces deux modificateurs sont d'un intérêt limité puisque

PRINT X IF X > 0 est équivalent à IF X > 0 THEN PRINT X.

11.8.3. Modificateur WHILE

L'exécution de l'instruction suivie de WHILE est répétée tant que la condition est satisfaite.

Exemple :
```
X = 1
X = 2 * X WHILE X < 17
```

X prendra successivement les valeurs 1, 2, 4, 8, 16 puis 32 et alors on passera à l'instruction suivante.

11.8.4. Modificateur UNTIL.

L'exécution de l'instruction suivie de UNTIL est répétée tant que la condition est fausse

Exemple :
```
X = 1
X = 2 * X UNTIL X > 17
```

X prendra comme précédemment les valeurs 1, 2, 4, 8, 16 puis 32 et alors la condition X > 32 étant satisfaite, on continuera en séquence.

11.8.5. Modificateur FOR.

Ce modificateur peut prendre l'une des formes syntaxiques rencontrées précédemment (cf. paragraphes 11.5 et 11.6). Dans chaque cas l'interprétation ne présente aucune difficulté à condition de se rappeler que l'instruction précédée de ce modificateur est assimilée à la portée d'une instruction FOR.

11.9. Exercices.

1. Soit un tableau A(100) chercher à partir du début du tableau, la valeur du plus grand élément en arrêtant les recherches dès que l'on rencontre un élément négatif.

Tracer l'organigramme et écrire le programme correspondant.

2. Écrire la séquence d'instructions permettant la résolution d'une équation selon la méthode de Newton à partir de l'organigramme simplifié suivant (organigramme meilleur donné en 1.2 mais à ne pas utiliser ici).

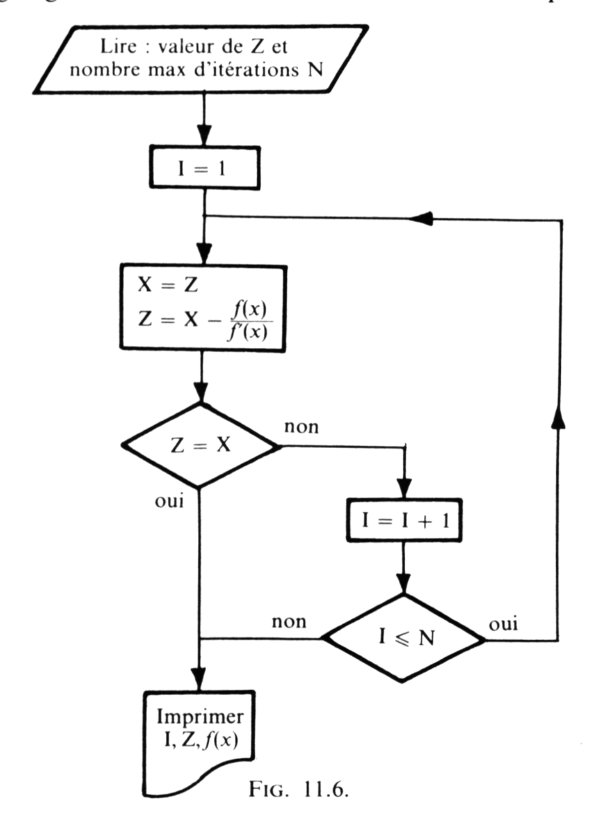


Fig. 11.6.

Réponses :

1. L'organigramme suivant peut convenir

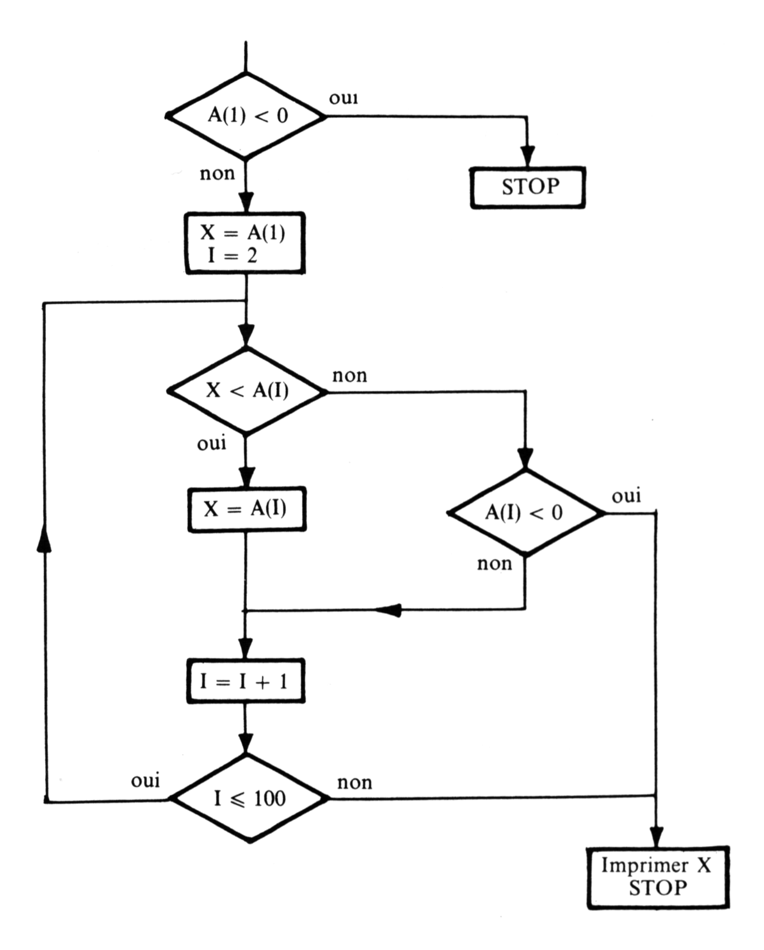


FIG. 11.7.

Une première séquence est la suivante :

```
0100 IF A(1)<P THEN STØP
0110 X=A(1)
0120 FØR I=2 TØ 100
0130    IF X<A(I) THEN X=A(I) ELSE IF A(I)<0 THEN 150
0140 NEXT I
0150 PRINT X
0160 STØP
```

2.

```
0100 REM FNA CØRRESPØND A LA FØNCTIØN ET FND A SA DERIVEE
0110 DEF FNA(X)= ...
0120 DEF FND(X)= ...
0130 INPUT Z,N
0140 FØR I=1 TØ N WHILE X<>Z
0150   X=Z
0160   Z=X-FNA(X)/FND(X)
0170 NEXT I
0180 PRINT I,Z,FNA(Z)
0190 END
```

11.10. Instructions d'entrées/sorties complémentaires.

L'allure de ces instructions varie beaucoup d'un constructeur à l'autre, en particulier pour les instructions de traitement de fichiers. Les extensions portent essentiellement sur les points suivants :

— Extension des instructions IMAGE :

— l'image peut être contenue dans une variable caractère, ce qui permet de la faire varier en cours d'exécution et d'avoir ainsi une « image dynamique »,

— l'image peut contenir des descripteurs variés, qui permettent une plus grande souplesse (cf. MARK II).

— Instruction PRINT IN FORM inspiré davantage des instructions d'entrées/sorties FORTRAN.

— Fonctions particulières de mise en page : par exemple, la fonction POS permet à l'utilisateur de connaître la position de la tête d'écriture sur son terminal ou sur un fichier (disponible sur les systèmes CDC-XDS-DEC).

11.10.1. Rôle des caractères utilisables dans une image.

Outre les caractères # et ↑ déjà rencontrés au chapitre 5, certains systèmes permettent d'utiliser le $ pour les variables numériques et les lettres L, R et C pour les variables caractères.

La lettre $ est utilisable sur les systèmes MARK II-CDC-XDS-DATA GENERAL.

```
0080 LET X=12
0090 PRINT USING 100,X,X,X
0100 :$###  $$$.##  $$$$
```

donnera à l'exécution

$ 12	$12.00	$12
1re zone	2e zone	3e zone

Le $ le plus à droite de la zone est imprimé, dans la mesure où il n'empiète pas sur la valeur à éditer.

Les lettres L, R et C (MARK II notamment) permettent de construire des zones pour variables caractères qui seront cadrées à gauche, à droite ou centrées.

La lettre E permet de décrire une zone qui est susceptible d'être allongée si la longueur de la chaîne le nécessite (dans ce cas il n'y a pas de troncation).

Pour les distinguer des caractères correspondants qui font partie des littéraux ordinaires, ces descripteurs doivent être précédés d'un caractère apostrophe.

Exemple :

```
0100: EXEMPLE 'EE ##.## 'LL 'RRRRR
0110 READ A$,B$,C$
0120 READ D
0130 PRINT USING 100,A$,D,B$,C$
0140 DATA ABCDEF,GHIJ,KL,-3.1415

EXEMPLE ABCDEF -3.14 GH    KL
```

11.10.2. Présentation des tableaux.

L'instruction de sortie des tableaux bénéficie également de possibilités de mise en page.

Constituons par exemple un tableau de nombres aléatoires (la fonction « Random » RND fournit des nombres aléatoires compris entre 0 et 1) :

```
0010 DIM A(3,10)
0020 FØR I=1 TØ 3
0030 FØR J=1 TØ 10
0040   LET A(I,J)=INT(100*RND(1))
0050 NEXT J
0060 NEXT I
0070 MAT PRINT USING 80,A
0080:### ### ### ### ### ### ### ### ### ###
0090 END

64  85  61   1  17  77  63  41  71   2

13  29  91   5  41  74  87  14  82  50

31  86  56  75  41  48  14  20  40  93
```

11.10.3. Descripteurs particuliers et leur emploi pour les fichiers.

Les compilateurs BASIC présentent des différences importantes quant à la nature et aux fonctions des descripteurs.

Le système XDS-940, par exemple, dispose d'une grande variété de descripteurs qui facilitent les manipulations de fichiers (l'énumération qui suit, non limitative, permet de comprendre dans le détail le programme proposé au paragraphe 10.5).

L'appel d'une instruction IMAGE utilise différents mots clés :

USING ou IN FØRM ou IN IMAGE.

Par exemple : 40 PRINT ... IN FØRM ...

Les fonctions de ces différents mots clés sont très voisines, et les descripteurs auxquels elles renvoient sont en général identiques.

La répétition des descripteurs par le compilateur facilite souvent l'écriture.

Ainsi : 100 A = " 3X 2X 3X "

A le même effet que : 100 A = " XXX XX XXX "

La différenciation de la nature des caractères d'une chaîne peut être réalisée à l'aide des descripteurs.

Ainsi :

- Le descripteur X accepte les caractères alphabétiques ou numériques.
- Le descripteur D accepte les caractères numériques seuls.
- Le descripteur A accepte les caractères alphabétiques seuls.

Le symbole / a des effets différents suivant qu'il s'applique à une entrée-sortie pour le terminal, ou pour un fichier.

Pour le terminal :

- en sortie il génère un Retour Chariot (abréviation CR),
- en entrée, il fait ignorer tous les caractères jusqu'au prochain CR.

Pour un fichier en longueur fixe :

- en sortie, il fait écrire un article complet, même si la longueur de la chaîne de caractères faisant partie de la liste est inférieure à la longueur d'article,
- en entrée, il fait ignorer tous les caractères jusqu'à la fin de l'article.

Le descripteur B

- en sortie génère un caractère « espace »,
- en entrée ignore un caractère.

Le descripteur R a des effets différents suivant qu'il s'applique à une entrée-sortie pour le terminal, ou pour un fichier.

Pour le terminal :

- en entrée, lit tous les caractères d'une chaîne (y compris le CR, sans toutefois l'affecter à la chaîne),

- en sortie, écrit les caractères d'une chaîne, suivis d'un caractère CR.

Pour un fichier en longueur fixe :

- en entrée, lit les caractères d'un article, à partir de la position actuelle du fichier, et les affecte à la chaîne,
- en sortie, écrit la chaîne à la suite des caractères du même article.

11.11. Opérations sur les nombres complexes.

En plus des variables et constantes numériques réelles et « caractères » certains systèmes admettent les variables et constantes complexes.

11.11.1. Rappel des définitions.

Un nombre complexe z en coordonnées cartésiennes et de la forme :

$$z = a + ib \qquad \text{avec} \qquad i^2 = -1$$

a est la partie réelle de z et b sa partie imaginaire.

Dans le plan complexe, z est représenté par un vecteur

On peut aussi écrire en coordonnées polaires :

$$z = \rho\,(\cos\varphi + i\sin\varphi) \text{ avec}$$

$$\rho = \sqrt{a^2 + b^2}$$

$$\text{tg}\,\varphi = \frac{b}{a}$$

ρ est le « module » de z et φ son « angle polaire » ou sa « phase ».

FIG. 11.8.

11.11.2. Instructions pour les constantes et variables complexes.

Pour que le compilateur réserve deux emplacements de mémoire pour un nombre complexe, il faut le déclarer comme tel au préalable.

Exemple : 10 CØMPLEX S, V(20), T(8, 4)

définit :
- une variable complexe S
- une variable complexe indicée V,
- un tableau de nombres complexes T à 8 lignes et 4 colonnes.

Pour les entrées-sorties, deux valeurs sont exigées pour chaque nombre complexe ; elles sont affectées, la première à la partie réelle, la deuxième à la partie imaginaire.

Exemple :

```
0010 CØMPLEX A,B
0020 READ A,B
0040 END
0050 DATA 0.8,1.4,7.6,-2
```

Dans cet exemple, on a :

$$A = 0{,}8 + 1{,}4i \qquad \text{et} \qquad B = 7{,}6 - 2i.$$

Les expressions arithmétiques pour les nombres complexes s'écrivent de la même manière que pour les autres variables.

Il y a lieu de noter toutefois que pour l'opération de comparaison, certains compilateurs donnent le résultat de la comparaison sur les parties réelles seules.

11.11.3. Fonctions complexes.

Le tableau qui suit présente les fonctions standard les plus courantes.

Z, Y et V sont des variables complexes.

Désignation	*Exemple*	*Effet de la fonction*
CMPLX(A, B)	20 Z = CMPLX(2, 3 + P)	affecte à z la valeur : $z = 2 + (3 + p)i$
REAL(X)	30 PRINT REAL(Z)	fournit la partie réelle de l'argument : ici le terminal imprime 2
IMAG(X)	40 PRINT IMAG(Z)	fournit la partie imaginaire de l'argument : ici le terminal imprime la valeur de $3+p$
CONJ	50 Y = CØNJ(Z)	fournit le nombre imaginaire conjugué de l'argument : ici $y = 2 - (3 + p)i$
PHASE(Z)	60 P = PHASE(Z)	fournit la phase (en radians) de l'argument
ABS(Z)	70 R = ABS(Z)	fournit le module de l'argument
PØLAR(R, P)	80 V = PØLAR(3, 0.2)	affecte à V les coordonnées polaires : 3 pour le module 0.2 radians pour la phase

11.11.4. Application : étude d'un circuit passif.

On considère le circuit ci-après :

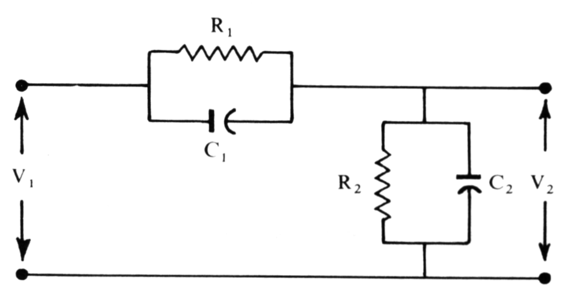


FIG. 11.9.

Il est demandé de calculer :

- le gain en tension (exprimé en décibel),
- le déphasage de la tension (exprimé en degré).

Les résultats doivent tenir compte des conditions suivantes :

- C_1 et C_2 sont fixés et exprimés en μF,
- R_1 prend les valeurs 0,2 MΩ et 0,8 MΩ.
- Pour chaque valeur de R_1, R_2 peut prendre les valeurs : 0,2 MΩ et 0,4 MΩ.
- Pour chaque couple de valeurs de R_1 et R_2, la pulsation W prend trois valeurs W_1 à W_3, telles que :

$$W_1 = 10 \text{ rad/s}$$
$$W_2 = W_1\sqrt{10} \text{ rad/s}$$
$$W_3 = W_2\sqrt{10} \text{ rad/s.}$$

On peut poser :

$$\frac{1}{Z_1} = \frac{1}{R_1} - \frac{C_1W}{i} \quad \text{et} \quad \frac{1}{Z_2} = \frac{1}{R_2} - \frac{C_2W}{i}$$

d'où $$Z_1 = \frac{iR_1}{i - R_1C_1W} \quad \text{et} \quad Z_2 = \frac{iR_2}{i - R_2C_2W}$$

d'où $$\frac{V_2}{V_1} = \frac{Z_2}{Z_1 + Z_2} = \frac{R_2(1 + iR_1C_1W)}{R_1(1 + iR_2C_2W) + R_2(1 + iR_1C_1W)}.$$

Soit : $$G = \frac{V_2}{V_1}.$$

Le gain exprimé en décibel est : $G_2 = 20 \log_{10} |G|$.

Et le déphasage en radian est :

$$P = \text{Arc tg} \frac{\text{partie imaginaire de G}}{\text{partie réelle de G}}$$

Pour la clarté du listing, on pose

$$N_1 = 1 + iR_1C_1W$$
$$N_2 = 1 + iR_2C_2W.$$

Les unités choisies nous permettent d'exprimer les formules sans modification. Le tracé de l'organigramme est immédiat (fig. 11.10).

Le programme suivant a été écrit en utilisant différentes possibilités du BASIC étendu.

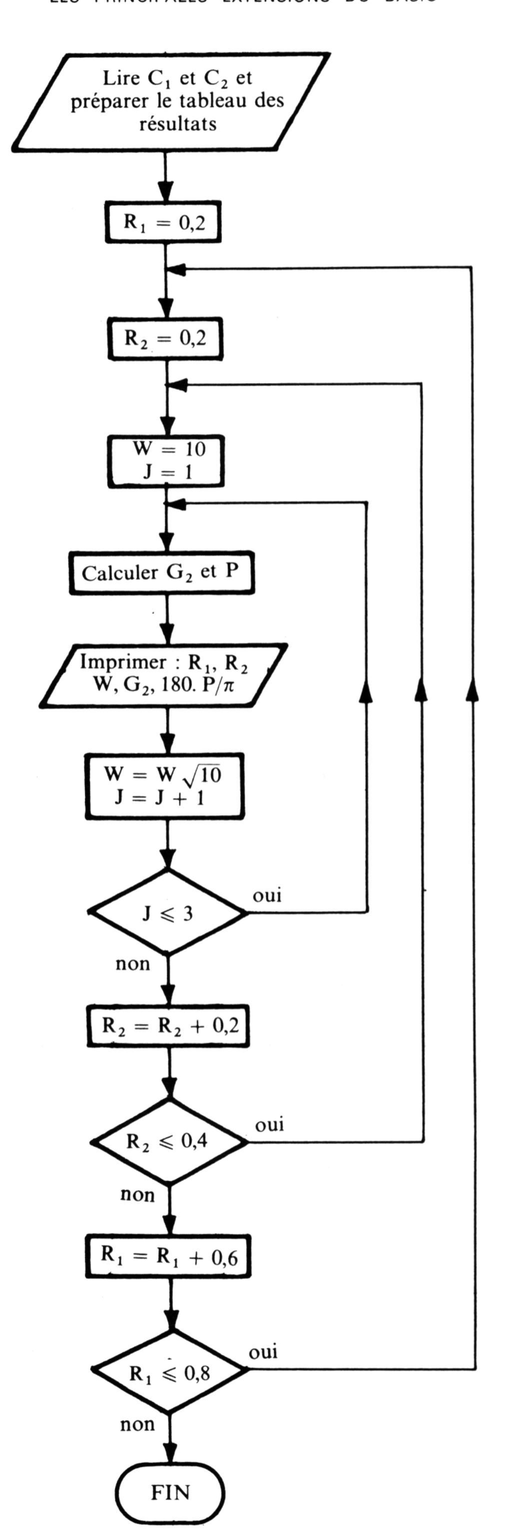
```
0100 CØMPLEX I,N1,N2,G
0110 READ C1,C2
0120 PRINT "VALEUR DE C1=",C1;"MICRØFARAD"
0130 PRINT "VALEUR DE C2=",C2;"MICRØFARAD"
0140 PRINT
0150 I=CMPLX(0,1)
0160 S=SQR(10)
0170 T="%.%    %.%    %%%%%.%%    %%%.%%%    %%.%%"
0180 PRINT " R1     R2        W        GAIN(DB)   DEPHASAGE"
0190 PRINT
0200 FØR R1=0.2 TØ 0.8 STEP 0.6        !R1 RESISTANCE EN MEGØHM
0210   FØR R2=0.2 TØ 0.4 STEP 0.2      !R2 RESISTANCE EN MEGØHM
0220     W=10
0230     FØR J=1 TØ 3
0240       N1=1+I*W*R1*C1
0250       N2=1+I*W*R2*C2
0260       G=R2*N1/(R1*N2+R2*N1)
0270       G2=20*LØG10(ABS(G))
0280       P=ATN(IMAG(G),REAL(G))                    !DEPHASAGE EN RADIAN
0290       PRINT IN IMAGE T : R1,R2,W,G2,180*P/PI !DEPHASAGE EN DEGRE
0300     W=W*S
0310 NEXT J,R2,R1
0320 STØP
0330 DATA 0.5,0.2
```

```
VALEUR DE C1=    .5    MICRØFARAD
VALEUR DE C2=    .2    MICRØFARAD

 R1     R2        W        GAIN(DB)    DEPHASAGE

 .2     .2      10.00      -4.742      10.01
 .2     .2      31.62      -3.315       6.76
 .2     .2     100.00      -2.967       2.42
 .2     .4      10.00      -3.233       1.97
 .2     .4      31.62      -2.981       1.17
 .2     .4     100.00      -2.929        .40
 .8     .2      10.00      -5.205      27.72
 .8     .2      31.62      -3.229      11.25
 .8     .2     100.00      -2.954       3.67
 .8     .4      10.00      -3.755      14.14
 .8     .4      31.62      -3.018       5.09
 .8     .4     100.00      -2.932       1.63
```

Résultat de l'exécution



Lire C_1 et C_2 et préparer le tableau des résultats
$R_1 = 0,2$
$R_2 = 0,2$
W = 10
J = 1
Calculer G_2 et P
Imprimer : R_1, R_2
W, G_2, 180. P/π
$W = W\sqrt{10}$
J = J + 1
$J \leqslant 3$
oui
non
$R_2 = R_2 + 0,2$
$R_2 \leqslant 0,4$
oui
non
$R_1 = R_1 + 0,6$
$R_1 \leqslant 0,8$
oui
non
FIN

FIG. 11.10.

11.12. Déclarations de chaînes DIM STRING et TEXT.

Au paragraphe 2.3.2 ainsi qu'au chapitre 8, la forme ainsi que la longueur maximum des constantes et des variables caractères ont été exposées.

Pour faciliter l'introduction des données et pour optimiser l'occupation de la mémoire existent des déclarations complémentaires.

11.12.1. Noms des variables et forme des données.

Le caractère $ qui précise la nature d'une variable comme étant une variable caractères n'est pas obligatoire sur le système XDS 940.

On pourra ainsi écrire indifféremment :

```
A$       ou   A
B3$      ou   B3
C$(3,2)  ou   C(3,2)
```

De plus, la présence d'apostrophes pour encadrer une constante de chaîne n'est plus obligatoire si le premier caractère est alphabétique. Par exemple :

```
70 DATA ALPHA, "2A * B"
```

11.12.2. Déclaration STRING.

On peut s'affranchir de la contrainte des guillemets, quel que soit le contenu de la constante de chaîne, en la déclarant comme telle par String. Par exemple :

```
10 STRING A, B, C(5)
70 DATA ALPHA, 2A * B, C1, C2, C3, C4, C5
```

Naturellement, quand un tableau est déclaré par STRING (c'est le cas de C ci-dessus), il doit contenir des chaînes exclusivement.

Quand il doit aussi contenir des nombres, il faut employer la déclaration DIM et introduire les constantes de chaînes encadrées par des guillemets.

11.12.3. Déclaration TEXT.

Par la déclaration TEXT, il est possible de limiter à sa valeur indispensable l'espace mémoire réservé par le compilateur pour les *tableaux* de chaînes.

Ainsi :

```
10 TEXT X(8) : 15, Y(2,3) : A * B
```

fait effectuer les réservations suivantes :

- pour X : 8 éléments de 15 caractères chacun au maximum ;
- pour Y : 6 éléments de A * B caractères au maximum.

11.13. Les subroutines avec transmission de paramètres.

Le système VARIAN présente une originalité très intéressante : la possibilité d'appeler des subroutines avec transmission de paramètres. Pour cela, il est fait appel à la déclaration SUB et à une forme particulière de l'instruction GOSUB.

11.13.1. L'instruction SUB.

Cette déclaration permet d'indiquer la liste des paramètres formels :

```
100   SUB X, Y, Z
200   Return
```

Ceci signifie que la séquence comprise entre les lignes 100 et 200 est un sous-programme.

11.13.2. L'instruction GOSUB.

Pour faire appel à l'instruction précédente, on écrira par exemple :

```
500 GOSUB   100, A, B, 3
```

Ceci signifie que lors du branchement, il faudra transférer les valeurs de A, B et de 3 respectivement aux paramètres formels X, Y, Z.

En outre, une telle subroutine peut être utilisée de façon récursive, c'est-à-dire s'appeler elle-même.

11.14. L'instruction Call.

Cette instruction existe sur différents systèmes, mais avec une signification différente :

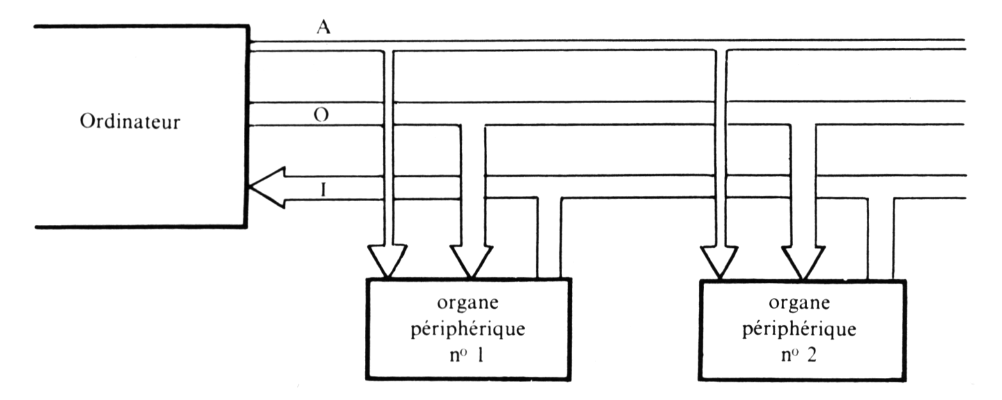
Télésystèmes	: permet d'appeler une fonction définie sur plusieurs lignes.
Varian	: permet d'appeler des subroutines externes.
Hewlett Packard	: permet d'appeler des subroutines écrites en assembleur.
Mark II	: permet d'appeler quelques subroutines particulières.

11.15. Extensions industrielles et instrumentales.

Dans les applications de type industriel, l'ordinateur est connecté à un système physique qui lui envoie des informations. Après un traitement qui

peut être simple (calcul de moyenne — comparaison à un seuil), l'ordinateur envoie des commandes.

Le schéma de telles installations est souvent le suivant :



La ligne d'adresse A sélectionne l'organe périphérique.

La ligne I est une ligne d'entrée d'informations.

La ligne O est une ligne de commande.

L'ensemble de ces 3 lignes forme un « Bus » ou « Omnibus ».

Pour ces applications, le BASIC comporte des extensions qui varient selon les ordinateurs. Par exemple pour le bus GP-IB, on aura des instructions dont la forme générale sera :

READ < adresse >, < liste >
WRITE < adresse >, < liste >

la longueur de la liste est limitée au nombre d'octets successifs transportés par le BUS.

Pour donner un délai de propagation suffisant entre ordinateur et périphérique, l'instruction WAIT est souvent disponible et peut prendre la forme suivante :

WAIT < expression >

où la valeur de l'expression indique le temps d'attente exprimé en millisecondes. Il existe d'autres formes pour cette instruction.

11.16. Extensions graphiques.

Le développement des ordinateurs de table à usage d'instrumentation (acquisitions de mesure, pilotage d'expériences, etc.) a conduit les constructeurs à proposer des extensions qui permettent la présentation de résul-

tats sous forme graphique par exemple des tracés de courbes. Ces instructions se classent en trois catégories :

— environnement, mise en page,
— entrées graphiques,
— sorties graphiques.

Parmi les constructeurs, Hewlett-Packard et Tektronix présentent de nombreuses instructions dont les lignes suivantes ne donnent qu'un aperçu.

Mise en page et environnement

Cadre	WINDOW	XMIN, XMAX, YMIN, YMAX	(1)
	LIMIT	XMIN, XMAX, YMIN, YMAX	(2)
Facteur d'échelle	SCALE	I, J	(1)
Tracé des axes	AXIS	I, J, K, L	(1)
	AXES	I, J, K, L, M, N	(2)

I et J graduent les axes
K et L positionnent l'origine
M et N marquent les grandes graduations

Sortie

MOVE XY	déplacement sans tracé d'une distance horizontale X et verticale Y ; par le premier	(1) (2)
RMOVE XY	X et Y représentent les coordonnées et pour le second X et Y représentent l'ancienne par rapport aux coordonnées du point précédent.	(1)

Ces déplacements se font sans tracé.

DRAW XY	déplacements similaires aux précédents	(1) (2)
RDRAW XY	mais avec tracé.	(1)
RPLOT X, Y, 1	déplacement relatif sans tracé	(2)
RPLOT X, Y, 0	déplacement relatif avec tracé	(2)

Entrée

GIN X, Y lit les coordonnées X et Y du point marqué par le réticule qui est déplacé manuellement sur l'écran. (1)

CURSOR X, Y même fonction que GIN (2)

11.17. Extensions pour « home computers »

Les microordinateurs à usage personnel ou familial sont souvent conçus en vue d'utiliser un téléviseur ordinaire comme écran pour afficher des

(1) Forme Tektronix.
(2) Forme Hewlett-Packard.

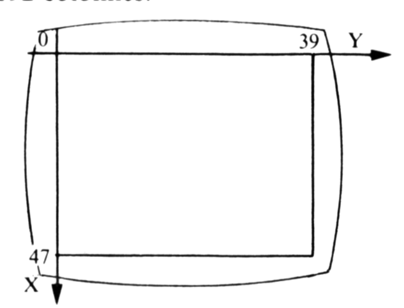
résultats. Certains constructeurs ont donc tiré parti de certaines caractéristiques du téléviseur pour offrir des possibilités graphiques pour l'utilisateur.

Enfin ces matériels disposent souvent du bus S100 qui nécessite des instructions particulières pour sa mise en œuvre.

11.17.1. Utilisation du téléviseur en mode graphique.

En mode normal l'écran est divisé en 48 lignes et 40 colonnes déterminant au total 1920 petits rectangles qui sont individuellement adressables.

En mode « haute résolution on peut obtenir jusqu'à 280 lignes et 192 colonnes.



Passage en mode graphique. Pour passer du mode alphanumérique en mode graphique il faut exécuter l'instruction :

GR ou GRAPHICS

ou pour la haute résolution :

HGR (1) ou HGR2 (1)

Choix de la couleur : certains matériels comme l' « APPLE » permettent de sélectionner une couleur au moyen de l'instruction :

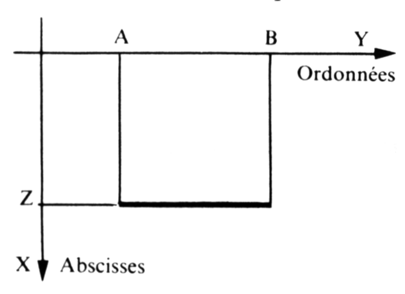
COLOR = expression ou HCOLOR = exp

où la partie entière de l'expression indique le numéro de la couleur sélectionnée parmi les suivantes (voir tableau p. 169).

Instructions graphiques : elles diffèrent selon le mode utilisé et nous ne citons pas toutes les instructions disponibles.

Les instructions PLOT X, Y en mode normal et HPLOT X, Y en mode haute résolution colorent la case de coordonnée X, Y.

L'instruction HLIN A, B, AT Z trace un segment horizontal porté par la droite d'abscisse Z et qui est limité par les ordonnées A et B. Comme précédemment on doit avoir A et B compris dans l'intervalle (0,39) et Z dans l'intervalle (0,47).



Dans la pratique, ce segment est obtenu par la coloration des petits rectangles d'abscisses Z et dont les ordonnées vont de A à B.

(1) Cas du microordinateur APPLE

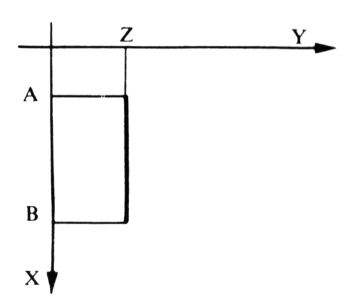
Numéro	*Mode normal*	*Mode haute résolution*
	COLOR :	HCOLOR :
0	noir	noir 1
1	magenta	vert
2	bleu foncé	bleu
3	violet (pourpre)	blanc 1
4	vert foncé	noir 2
5	gris	dépendent des téléviseurs
6	bleu	dépendent des téléviseurs
7	vert	blanc 2
8	marron	
9	orange	
10	gris	
11	rose	
12	vert	
13	jaune	
14	bleu	
15	blanc	

Couleur par défaut : noir

De même on tracera une ligne verticale au moyen de l'instruction :

VLIN A, B AT Z

dans laquelle Z représente l'ordonnée et A et B les abscisses début et fin.



11.17.2. Utilisation du bus S100

Le bus S100 est un bus standard des ordinateurs amateurs qui permet la connexion de différents périphériques. On peut l'utiliser au moyen d'instructions particulières.

A titre d'exemple le bus S100 du microordinateur « SORCERER » permet la connexion de minidisquettes, d'un générateur de sons, d'une

unité de « réponse vocale », d'un interface de transmission 110/300 bauds, etc.

La commande de ce bus se fait notamment au moyen de l'instruction OUT et de la fonction INP.

OUT I,J

envoie la valeur J (qui doit tenir sur un octet) sur la ligne I (périphérique n° I).

La fonction INP (I) permet de lire l'octet qui vient de la ligne I. On écrira :

X = INP (I)

11.17.3. Instruction POKE – fonction PEEK

Pour permettre l'échange d'informations entre un programme BASIC et un sous-programme écrit dans un autre langage (langage d'assemblage en général), de nombreux BASIC disponibles sur micro-ordinateurs mettent à la disposition des utilisateurs l'instruction POKE et la fonction PEEK.

Instruction POKE — Cette instruction dont la forme est

POKE I, J

permet d'écrire dans la mémoire d'adresse I, la valeur numérique contenue dans la variable ou l'expression J. Cela suppose que I représente une valeur entière inférieure à l'adresse maximale disponible et J une valeur qui puisse tenir à cette adresse (de 0 à 255 s'il s'agit d'adresse d'octet).

Fonction PEEK — Cette fonction dont la forme est

PEEK I

donne le contenu de la mémoire dont l'adresse est donnée par la valeur de I qui doit satisfaire aux mêmes conditions que précédemment.

Certains BASIC (MICROSOFT par exemple) acceptent l'utilisation de constantes hexadécimales qui sont très commodes pour la mise en œuvre de PEEK et POKE.

CHAPITRE 12

EXERCICES DE RÉCAPITULATION

12.1. Énoncés.

12.1.1. Calcul du nombre d'or à partir d'une série.

La valeur de ce nombre peut être obtenue à partir du calcul suivant :

$$\left.\begin{aligned} u_1 &= 1 \\ u_2 &= 2 \\ u_n &= u_{n-1} + u_{n-2} \end{aligned}\right\} \quad \text{suite de Fibonacci.}$$

La suite V_n définie par

$$v_n = \frac{u_n}{u_{n-1}} \quad \text{tend vers le nombre d'or}$$

Écrire un programme qui calcule et imprime v_n pour n variant de 3 à 20.

12.1.2. Calcul des nombres parfaits.

Un nombre entier est dit parfait s'il est égal à la somme de ses diviseurs, 1 compris et lui-même non compris.

Exemple : 6 est parfait car $6 = 1 + 2 + 3$.

Écrire l'organigramme puis le programme permettant à partir de deux données M et N d'imprimer les nombres parfaits situés dans l'intervalle [M, N].

12.1.3. Table de Pythagore.

Écrire un programme permettant d'obtenir l'édition d'une table de Pythagore 10 × 10.

Rappelons qu'une table de Pythagore est un tableau 10 × 10 tel que :

$$A(I, J) = I * J$$

12.1.4. Volume d'une cuve cylindrique.

Soit une cuve cylindrique (cuve à mazout par exemple) posée horizontalement. Établir un programme dressant une table indiquant le volume contenu dans le réservoir en fonction de la hauteur *h*.

Les données à lire sont dans l'ordre :

— le rayon du cylindre R
— la longueur de la cuve L
— le pas dans la table P

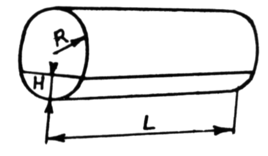


FIG. 12.1.

NOTA : Pour ce problème il est indispensable de dessiner l'organigramme avant de commencer l'écriture du programme.

12.1.5. Équation du second degré.

— Écrire la séquence d'instructions correspondant à l'organigramme de la figure 1.1.

— Dessiner un nouvel organigramme permettant de traiter le cas de racines complexes et écrire le programme correspondant.

12.1.6. Calcul de mensualités.

Un capital est emprunté à un taux d'intérêt annuel donné. Il est remboursé par des mensualités constantes dont le nombre est fixé par l'emprunteur.

Le programme détermine en particulier :

- la valeur de la mensualité,
- les parts d'intérêt et de remboursement de capital comprises dans chaque mensualité.

Le « taux équivalent » T_1 est d'abord calculé. T_1 est le taux mensuel qui, appliqué au même capital que le taux annuel T, produit au bout d'un an le même intérêt.

$$(1 + T_1)^{12} = 1 + T$$

ou

$$T_1 = (1 + T)^{1/12} - 1$$

Si N est le nombre de mensualités choisies, et C le capital emprunté, la valeur de la mensualité est :

$$M = \frac{CT_1}{1 - (1 + T_1)^{-N}}.$$

Les données introduites par l'opérateur sont les suivantes :

- capital emprunté,
- taux annuel,
- nombre de mensualités choisi pour le remboursement.

Les résultats produits sont les suivants :

- valeur de la mensualité,
- total des remboursements,
- taux équivalent,
- parts d'intérêt et de remboursement de capital comprises dans chaque mensualité.

12.1.7. Rentabilité d'un investissement.

Un entrepreneur doit étudier la rentabilité prévisionnelle d'un investissement, dont il connaît le montant. Le taux annuel de l'argent est également une donnée. Il a par ailleurs estimé les charges et les profits annuels attendus sur la durée de vie de l'investissement ainsi que sa valeur de revente sur le marché d'occasion.

Le programme détermine en particulier :

- le bénéfice actualisé,
- le taux de rentabilité immédiate,
- le délai de récupération,
- le taux interne de rentabilité.

Les données sont définies de la manière suivante :

C	coût initial de l'investissement
V	durée de vie
T	taux d'actualisation
RO	valeur résiduelle de revente (peut être nulle)
R(I)	recette attendue pour l'année I
D(I)	dépense attendue pour l'année I

Avec ces notations, on obtient :

bénéfice de l'année I : B(I) = R(I) − D(I)
bénéfice de l'année I, actualisé : B1(I) = B(I) * (1 + T)↑ (− I)

On suppose par ailleurs que l'investissement est revendu dans l'année qui suit la fin de durée de vie.

Le bénéfice total actualisé est donc :

$$\mathrm{BO} = \sum_{I=1}^{I=V} \mathrm{B1(I)} + \mathrm{RO} * (1 + \mathrm{T})\uparrow(-\mathrm{V}-1) - \mathrm{C}.$$

Le délai de récupération J se définit comme la durée limite, si elle existe, qui rend BO positif ou nul.

Le taux de rentabilité immédiate a pour valeur : B(1)/C.

Le programme calcule enfin, seulement quand BO $\geqslant$ 0, le taux interne de rentabilité TO.

TO est le taux d'actualisation limite qui rend BO nul. TO est solution de l'équation :

$$0 = \sum_{I=1}^{I=V} B(I) * (1 + TO) \uparrow (-I) + RO * (1 + TO) \uparrow (-V-1) - C.$$

REMARQUE : Le lecteur intéressé par ce sujet pourra consulter « L'entreprise face à la décision d'investir ». La Documentation Française.

12.1.8. Résolution d'un système linéaire par la méthode de Gauss.

A étant une matrice carrée, de dimensions $n \times n$, nous désirons résoudre le système linéaire AX = B selon la méthode de GAUSS.

Cette méthode comporte deux étapes :

— se ramener à un système dont la matrice est triangulaire (par exemple triangulaire supérieure) :

$$TX = C,$$

— résoudre ce système.

Passage à une matrice triangulaire : le principe consiste par des combinaisons de lignes à faire apparaître des zéros au-dessous de la diagonale principale.

Si $a_{11} \neq 0$, on élimine tous les termes a_{i1} pour $i \neq 1$ par combinaison linéaire entre la ligne 1 et la ligne i et ceci pour i variant de 2 à n.

Exemple : Soit à résoudre

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 5 \\ 0 & 3 & 0 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ X_2 \end{pmatrix} = \begin{pmatrix} 7 \\ 11 \\ 3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 0 & 2 \\ 0 & 3 & 0 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \\ 3 \end{pmatrix}$$

Ensuite si $a_{22} \neq 0$, on élimine tous les termes a_{i2} pour i variant de 3 à n toujours par combinaison linéaire entre la ligne 2 et les lignes suivantes.

On poursuivra l'élimination jusqu'à ce que la matrice soit triangulaire supérieure.

Le terme a_{ii} qui sert à annuler les éléments situés au-dessous de lui est appelé PIVOT. Il peut arriver qu'un pivot soit nul. C'est précisément le cas de l'exemple avec a_{22}. Dans ce cas, on cherchera un élément a_{hi} pour $h \neq i$ qui soit différent de zéro et on permutera les lignes h et i puis on reprendra l'opération de triangularisation au point où elle a été interrompue.

Reprenons l'exemple précédent : par permutation des lignes 2 et 3, nous obtenons :

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 3 \\ 4 \end{pmatrix}$$

Il se trouve ici par chance que l'opération de triangularisation est terminée. Si en fin de triangularisation $a_{nn} = 0$, la matrice est singulière et la résolution est à abandonner.

Résolution du système triangulaire : la détermination de x_n est immédiate :

$$x_n = \frac{C_{nn}}{t_{nn}}$$

pour obtenir la valeur de x_{n-1}, il suffira d'utiliser la valeur de x_n précédemment trouvée. D'une manière générale, nous obtiendrons x_i par :

$$x_i = \frac{1}{t_{ii}}\left(C_i - \sum_{j=i+1}^{n} t_{ij}x_j\right).$$

L'exemple précédent donne :

$$x_3 = \frac{4}{2} \qquad\qquad = 2$$

$$x_2 = \frac{1}{3}(3 - 0 \times 2) \qquad = 1$$

$$x_1 = \frac{1}{1}(7 - 2 \times 1 - 3 \times 2) = 1.$$

Problème :

— écrire l'organigramme d'un programme faisant une telle résolution sans utiliser les instructions matricielles de calcul,

— la matrice triangulaire sera construite à la place même de la matrice A,

— écrire ensuite le programme BASIC correspondant. La lecture des données pourra se faire par l'instruction MAT READ.

12.1.9. Problème de minimisation de câblage (exercice difficile).

Soit à réunir *n* points de façon à minimiser le coût total. On dispose d'une matrice symétrique (avec diagonale nulle) dont l'élément c_{ij} indique le coût de la liaison du point *i* au point *j*.

i \ j	1	2	3	4	5	6	7	8	9
1	0								
2		0							
3			0						
4				0					
5					0				
6						0			
7							0		
8								0	
9									0

FIG. 12.2.

La méthode connue sous le nom d'Algorithme de Kruskal est la suivante :

— choisir parmi les *n* points, les deux points λ_1 et λ_2 tels que $C_{\lambda 1, \lambda 2}$ soit le plus petit.

Ces deux points constituent l'ensemble $\Omega 1$ et on réunit λ_1 et λ_2, l'exemple fera ainsi réunir les points 5 et 6.

— déterminer le point λ_3 tel que sa distance à $\Omega 1$ soit minimum. On doit donc chercher *i* tel que :

$$\text{Min}\,(C_{i,\lambda 1},\ C_{i,\lambda 2})$$
$$i = l, n$$
soit obtenue avec $i \neq \lambda_1,\ i \neq \lambda_2$.

Ce point λ_3 sera réuni à $\Omega 1$ pour donner l'ensemble $\Omega 2$; le point 2 de l'exemple sera le plus proche de $\Omega 1$ (composé des points 5 et 6) et sera donc constitué des points 2, 5 et 6.

On répète l'opération précédente sur $\Omega 2$ pour constituer $\Omega 3$ et ainsi de suite, jusqu'à ce que tous les points soient reliés.

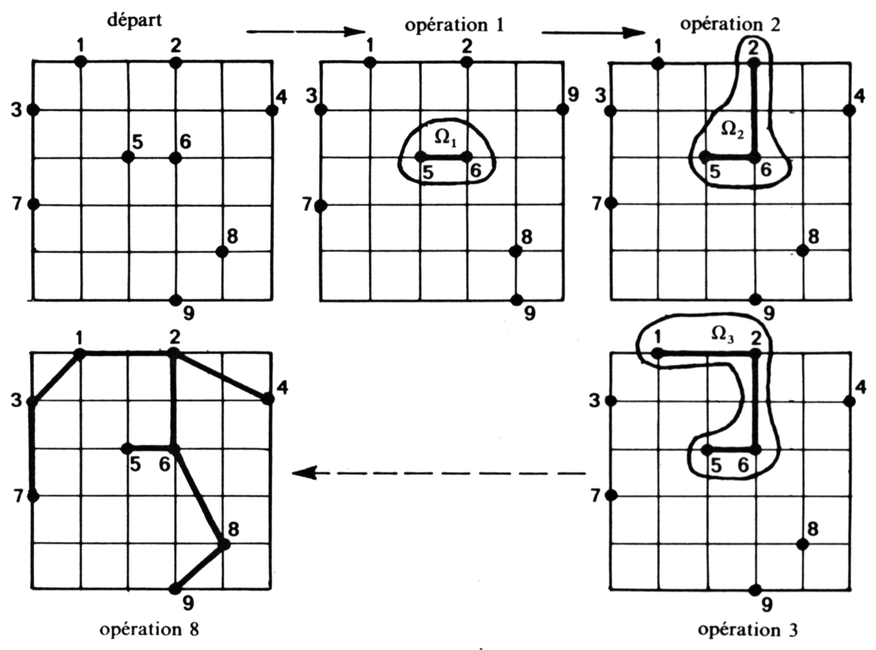


FIG. 12.3.

NOTA : la liaison 2-4 est équivalente à la liaison 6-4 ce qui montre que ce problème n'admet pas une solution unique.

Problème : dessiner l'organigramme et écrire le programme correspondant.

12.2. Solutions.

12.2.1. Nombre d'or.

Deux méthodes différentes sont possibles. L'organigramme de gauche (fig. 12.4) correspond à la méthode suggérée par la formulation mathématique. Celui de droite correspond à une économie de plan mémoire. Les programmes correspondants sont les suivants :

```
0010 DIM U(20)
0020 LET U(1)=1
0030 LET U(2)=2
0040 FØR I=3 TØ 20
0050   LET U(I)=U(I-1)+U(I-2)
0060   PRINT I,U(I)/U(I-1)
0070 NEXT I
0080 END
```

```
10 U=1
20 V=2
30 FOR I=3 TO 20
40   W=U+V
50   PRINT I;W/V
60   U=V
70   V=W
80   NEXT I
90 END
```

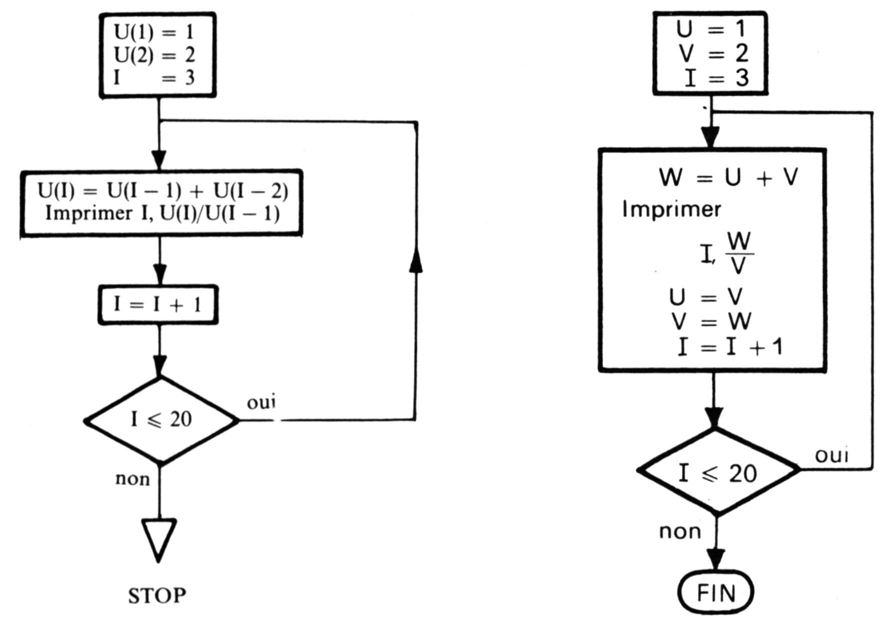


FIG. 12.4.

12.2.2. Nombres parfaits.

Pour savoir si un nombre est parfait, il suffit de chercher tous ses diviseurs et de faire leur somme.

Si cette somme est égale au nombre, alors ce nombre est parfait et il est imprimé.

L'organigramme suivant ne permet pas d'obtenir le nombre 1 bien que celui-ci soit parfait.

Pour savoir si H est diviseur de I, on peut en BASIC effectuer le calcul suivant :

$$R = I - \text{entier}\,(I/H) \times H.$$

Si R est nul alors H est un diviseur de I. La fonction « entier de » existe en BASIC sous le nom de INT.

On peut aussi comparer I/H et INT(I/H). Si ces deux quantités sont égales, alors H est un diviseur de I ainsi que le quotient.

Au niveau de l'organigramme, on peut distinguer deux solutions :

La première solution consiste à chercher des diviseurs situés entre 2 et I/2.

La seconde solution consiste à chercher les diviseurs situés entre 2 et $\sqrt{I}$ et à effectuer le cumul avec le quotient obtenu. Cette dernière méthode est évidemment plus rapide.

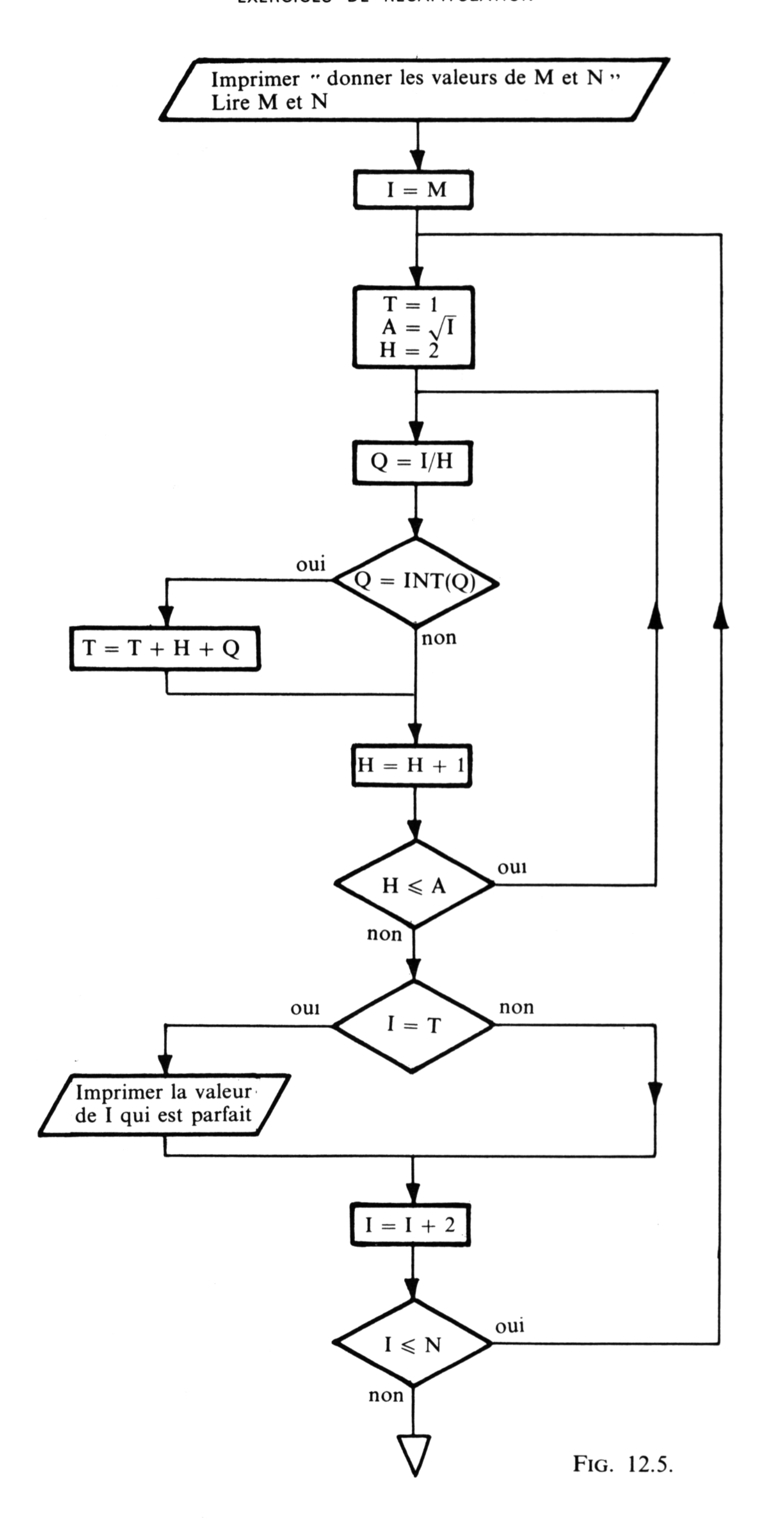

Imprimer " donner les valeurs de M et N "
Lire M et N
I = M
T = 1
A = √I
H = 2
Q = I/H
Q = INT(Q)
oui
non
T = T + H + Q
H = H + 1
H ⩽ A
oui
non
I = T
oui
non
Imprimer la valeur de I qui est parfait
I = I + 2
I ⩽ N
oui
non

FIG. 12.5.

Nous ferons l'hypothèse que les nombres parfaits sont tous pairs. L'écriture de la séquence correspondante est alors évidente.

```
0010 PRINT "DØNNER LES VALEURS DE M ET N :";
0020 INPUT M,N
0040 FØR I=M TØ N STEP 2
0050   LET T=1
0070   FØR H=2 TØ SQR(I+1)
0080     LET Q=INT(I/H)
0085     LET R=I-Q*H
0090     IF R<>0 THEN 110
0100     LET T=T+Q+H
0110   NEXT H
0120   IF I<> T THEN 140
0130   PRINT I;
01TO NEXT I
0150 END
```

```
DØNNER LES VALEURS DE M ET N : 494,500
  496
STØP AT LINE  0150
```

REMARQUE :

1. Les nombres parfaits sont très rares ; on peut citer néanmoins : 1, 6, 28, 496, 8128.

2. Ce programme serait d'écriture plus simple et d'exécution plus rapide à partir d'un BASIC acceptant des variables entières.

12.2.3. Table de Pythagore.

L'organigramme se trace sans difficulté à condition de penser qu'il y aura lieu de faire progresser l'indice ligne et l'indice colonne ; deux solutions sont envisageables (fig. 12.6 et 12.7) :

Les programmes correspondants sont les suivants :

```
0100 FØR I=1 TØ 10
0110 FØR J=1 TØ 10
0120 PRINT I*J;
0130 NEXT J
0140 PRINT
0150 NEXT I
0160 END
```

```
0100 FØR I=1 TØ 10
0110 FØR J=1 TØ 10
0120 A(I,J)=I*J
0130 NEXT J
0140 NEXT I
0150 MAT PRINT A;
0160 END
```

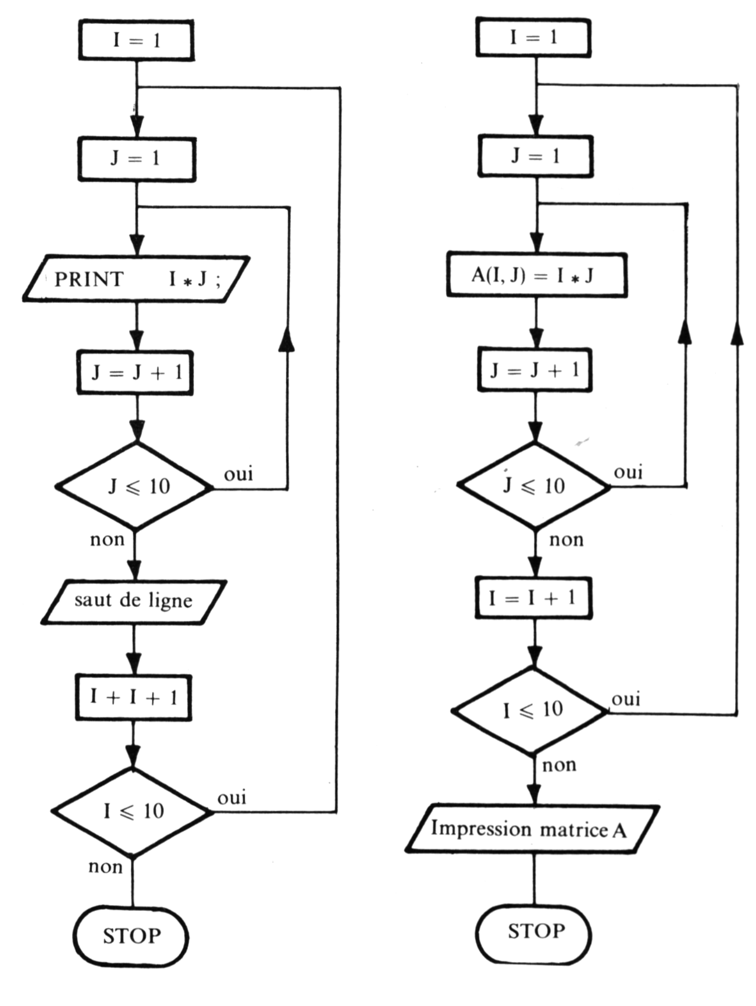


FIG. 12.6. FIG. 12.7.

12.2.4. Volume d'une cuve cylindrique.

La surface hachurée correspond à un segment de cercle, dont la surface est donnée ici par

$$S = R^2\theta - OI \,.\, IB$$

avec

$$OI = R - h$$

$$IB = \sqrt{R^2 - (R - h)^2}$$

$$= \sqrt{h(2R - h)}$$

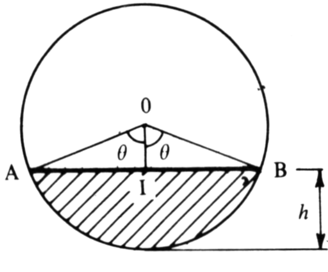


Fig. 12.8.

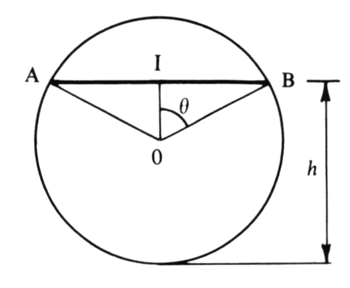


Fig. 12.9.

et par conséquent

$$\theta = \text{arctg}\left[\frac{\sqrt{h(2R-h)}}{R-h}\right]$$

nous pouvons donc écrire

$$S = R^2 \text{ arctg}\left[\frac{\sqrt{h(2R-h)}}{R-h}\right] - (R-h)\cdot\sqrt{h(2R-h)}$$

Dans le cas où la hauteur h est supérieure au rayon nous écrirons

$$\theta = \text{arctg}\left[\frac{\sqrt{h(2R-h)}}{R-h}\right] + \pi$$

puisque la fonction arctangente du BASIC nous fera obtenir un angle situé dans le quadrant $0, \frac{\pi}{2}$.

Le calcul du volume est ensuite immédiat :

$$V = S.L.$$

Afin d'éviter de recalculer certaines quantités, nous poserons :

$$E = R^2$$
$$D = 2R$$
$$A = \sqrt{H(D-H)}$$
$$B = R - H$$
$$C = A/B.$$

L'organigramme est alors le suivant :

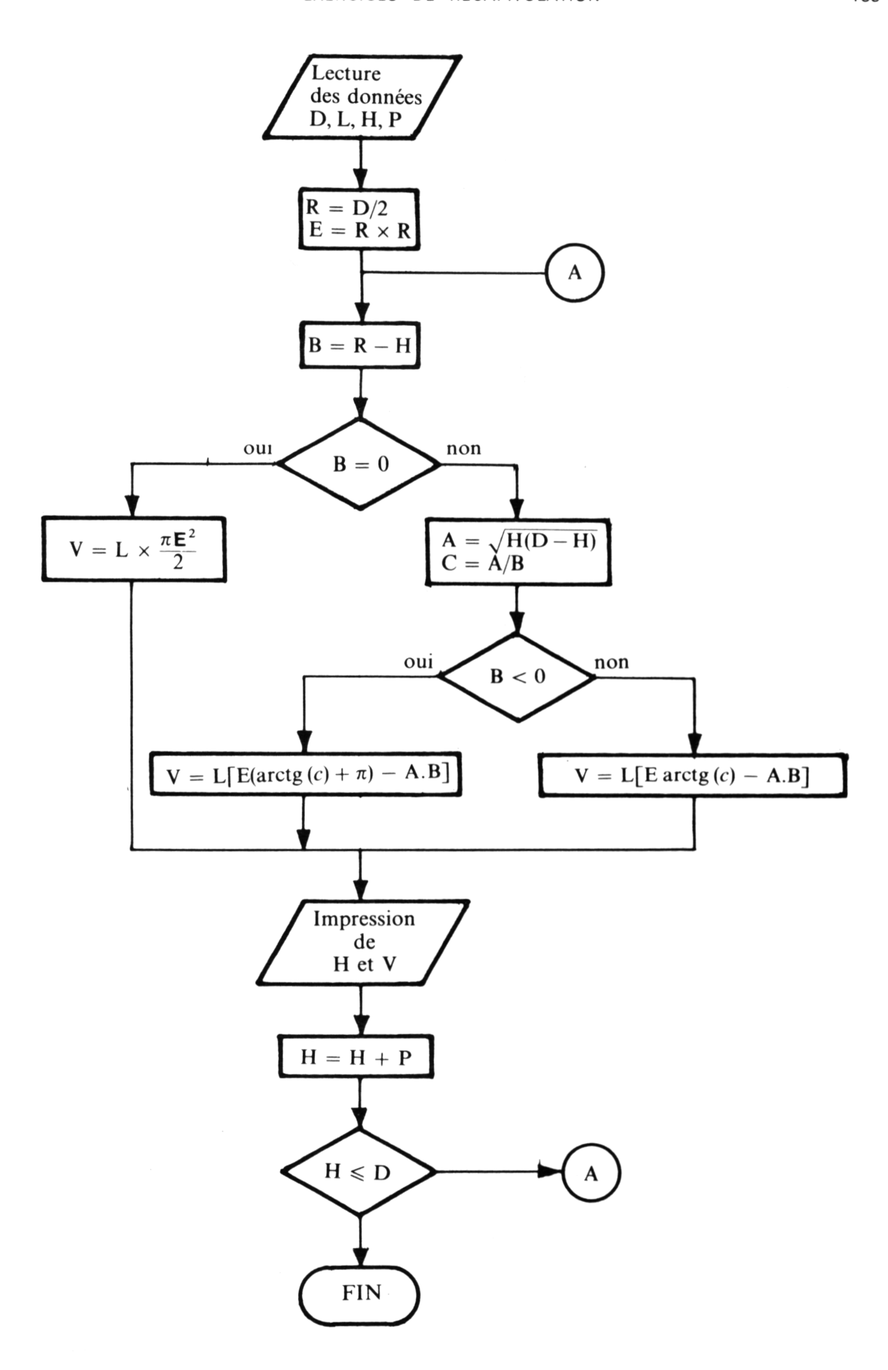

Lecture
des données
D, L, H, P
R = D/2
E = R × R
A
B = R − H
oui
B = 0
non
V = L × πE²/2
A = √H(D − H)
C = A/B
oui
B < 0
non
V = L[E(arctg (c) + π) − A.B]
V = L[E arctg (c) − A.B]
Impression
de
H et V
H = H + P
H ⩽ D
A
FIN

FIG. 12.10.

L'écriture du programme est alors évidente.

```
0002 LET PI=3.14159265
0005 PRINT"DØNNER DANS L'ØRDRE, EN METRES, LES VALEURS "
0006 PRINT"  DU DIAMETRE DU CYLINDRE,"
0007 PRINT"  DE SA LØNGUEUR"
0008 PRINT"  DE LA HAUTEUR INITIALE DE LIQUIDE,"
0009 PRINT"  DU PAS D'INCREMENTATIØN"
0010 PRINT
0015 PRINT"EXEMPLE: 0.5,3.,0.,0.05"
0020 PRINT
0025 INPUT D,L,H,P
0030 PRINT
0035 PRINT USING 105
0040 LET R=D/2
0045 LET E=R*R
0050 LET A=SQR(H*(D-H))
0055 LET B=R-H
0060 IF B<>0 THEN 75
0065 LET V=L*PI*E/2
0070 GØ TØ 100
0075 LET C=A/B
0080 IF B<=0 THEN 95
0085 LET V=L*(E*ATN(C)-A*B)
0090 GØ TØ 100
0095 LET V=L*(E*(ATN(C)+PI)-A*B)
0100 PRINT USING 110,H,V
0105 : HAUTEUR           VØLUME
0110 :  ###.##         #####.###
0115 LET H=H+P
0120 IF H<=D THEN 50
0125 PRINT
0130 PRINT TAB(10), "FIN"
0135 END
```

REMARQUE : En FORTRAN, la fonction ATAN2 permet de confondre les cas $h \leqslant R$ et $h > R$ et de traiter le problème de façon beaucoup plus élégante.

12.2.5. Équation du second degré.

L'organigramme de la figure 1.1 permet d'écrire rapidement la séquence suivante :

```
0100 INPUT A,B,C
0110 D=B↑2-4*A*C
0120 IF D<0 THEN 200
0130 IF D=0 THEN 170
0140 X1=(-B-SQR(D))/(2*A)
0150 X2=(-B+SQR(D))/(2*A)
0160 GØ TØ 180
0170 X1=X2=-B/(2*A)
0180 PRINT "X1 =",X1;"  X2 =",X2;
0190 STØP
0200 PRINT "PAS DE RACINE REELLE"
0210 END
```

Une autre technique consiste à utiliser la fonction SGN(X) dont la valeur permet d'effectuer l'aiguillage de façon élégante.

```
0100 INPUT A,B,C
0110 D=B↑2-4*A*C
0120 ØN SGN(D) GØ TØ 140,200,220
0130 REM   CAS DE RACINES CØMPLEXES
0140 PRINT "RACINES CØMPLEXES"
0150 PRINT "X1: PARTIE REELLE     =",-B/(2*A)
0160 PRINT "    PARTIE IMAGINAIRE=",-SQR(-D)
0170 PRINT "X2: PARTIE REELLE     =",-B/(2*A)
0180 PRINT "    PARTIE IMAGINAIRE=",SQR(-D)
0190 STØP
0200 PRINT "RACINES DØUBLES X1=X2=",-B/(2*A)
0210 STØP
0220 PRINT "X1=",(-B-SQR(D))/(2*A),"  X2=",(-B+SQR(D))/(2*A)
0230 END
```

Dans les deux cas les instructions PRINT peuvent être remplacées par des instructions PRINT USING et IMAGE.

12.2.6. Calcul de mensualités.

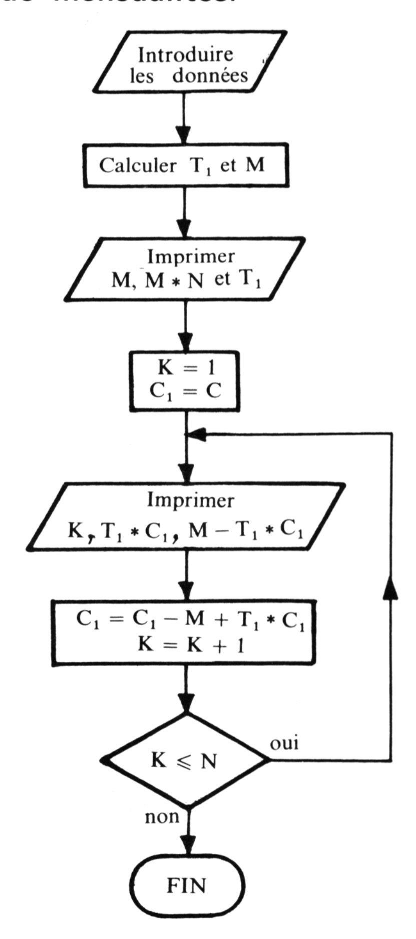


FIG. 12.11.

```
0010 PRINT "CAPITAL EMPRUNTE :       ",
0020 INPUT C
0030 PRINT "TAUX ANNUEL EN % :       ",
0040 INPUT T
0050 PRINT "NØMBRE DE MENSUALITES : ",
0060 INPUT N
0070 LET T1=(1+T/100)↑(1/12)-1
0080 LET M=C*T1/(1-(1+T1)↑(-N))
0090 PRINT USING 210,M
0100 PRINT USING 220,M*N
0110 PRINT USING 230,100*T1
0120 PRINT
0130 PRINT
0140 PRINT "NUMERØ       INTERET         CAPITAL"
0150 LET C1=C
0160 FØR K=1 TØ N
0170   PRINT USING 240,K,T1*C1,M-T1*C1
0180   LET C1=C1-M+T1*C1
0190 NEXT K
0200 END
0210:VALEUR DE LA MENSUALITE :     #####.##
0220:TØTAL DES REMBØURSEMENTS : #######.##
0230:TAUX EQUIVALENT EN % :          ##.####
0240:  ###        ####.##       ######.##
```

```
CAPITAL EMPRUNTE :          15000
TAUX ANNUEL EN % :          8
NØMBRE DE MENSUALITES :  20
VALEUR DE LA MENSUALITE :        801.70
TØTAL DES REMBØURSEMENTS :    16033.94
TAUX EQUIVALENT EN % :               .6434


NUMERØ       INTERET         CAPITAL
    1          96.51          705.19
    2          91.97          709.72
    3          87.41          714.29
    4          82.81          718.89
    5          78.19          723.51
    6          73.53          728.17
    7          68.85          732.85
    8          64.13          737.57
    9          59.38          742.31
   10          54.61          747.09
   11          49.80          751.89
   12          44.96          756.73
   13          40.10          761.60
   14          35.20          766.50
   15          30.26          771.43
   16          25.30          776.40
   17          20.30          781.39
   18          15.28          786.42
   19          10.22          791.48
   20           5.13          796.57
```

12.2.7. Rentabilité d'un investissement.

L'organigramme est le suivant :

Introduire les données :
C, V, T, R(I), R0, D(I)

J = V + 10
B = 0
Année I = 1

Calculer B(I)
et cumuler B

B < C
non
oui

J = 1

Imprimer : I, R(I),
D(I), B(I)

I = I + 1

I ⩽ V
oui
non

Imprimer : B(0)
et B(1)/C

J : : V
J > V
J < V
J = V

B0 ⩾ 0
non
oui

Imprimer :
« pas rentable »

Imprimer la
valeur V + 1

Imprimer la valeur
de V

Imprimer la valeur
de J

Imprimer T0

FIN

FIG. 12.12.

```
0010 DIM R(15),D(15),B(15),A(15)
0020 PRINT "CØUT INITIAL :",
0030 INPUT C
0040 PRINT "DUREE DE VIE :",
0050 INPUT V
0060 PRINT "TAUX D'ACTUALISATIØN EN % :",
0070 INPUT T
0080 PRINT "RECETTES ATTENDUES :",
0090 FØR I=1 TØ V
0100   INPUT R(I)
0110 NEXT I
0120 PRINT "VALEUR RESIDUELLE :",
0130 INPUT R0
0140 PRINT "DEPENSES ATTENDUES :",
0150 FØR I=1 TØ V
0160   INPUT D(I)
0170 NEXT I
0180 LET J=V+10
0190 LET B=0
0200 PRINT
0210 PRINT
0220 PRINT "ANNEE      RECETTES      DEPENSES      BENEFICES"
0230 PRINT "           ATTENDUES     ATTENDUES     ATTENDUS"
0240 PRINT
0250 PRINT
0260 FØR I=1 TØ V
0270   LET B(I)=R(I)-D(I)
0280   LET B=B+B(I)*(1+T/100)↑(-I)
0290   IF B<C THEN 310
0300   LET J=I
0310   PRINT USING 580,I,R(I),D(I),B(I)
0320 NEXT I
0330 LET B0=B+R0*(1+T/100)↑(-V-1)-C
0340 PRINT
0350 PRINT USING 600,B0
0360 PRINT USING 610,100*B(1)/C
0370 IF J<=V THEN 430
0380 IF B0>=0 THEN 410
0390 PRINT "L'INVESTISSEMENT PRØJETE N'EST PAS RENTABLE"
0400 GØ TØ 570
0410 PRINT USING 590,V+1
0420 GØ TØ 480
0430 IF J<V THEN 460
0440 PRINT USING 590,V
0450 GØ TØ 480
0460 PRINT USING 590,J
0470 IF B0<=0 THEN 570
0480 FØR T0=T TØ 100 STEP 0.1
0490   LET X=0
0500   FØR I= 1 TØ V
0510      LET X=B(I)*(1+T0/100)↑(-I)+X
0520   NEXT I
0530   LET X=X+R0*(1+T0/100)↑(-V-1)-C
0540   IF X<=0 THEN 560
0550 NEXT T0
0560 PRINT USING 620,T0
0570 END
0580: ###         ######        ######        ######
0590:LE DELAI DE RECUPERATIØN EST DE### ANNEES
0600:BENEFICE ACTUALISE : ######.##
0610:TAUX DE RENTABILITE IMMEDIATE###.## %
0620:TAUX INTERNE DE RENTABILITE###.## %
```

```
CØUT INITIAL : 60000
DUREE DE VIE : 4
TAUX D'ACTUALISATIØN EN % : 12
RECETTES ATTENDUES : 12000,17000,32000,22000
VALEUR RESIDUELLE : 5000
DEPENSES ATTENDUES : 6500,7000,9000,9500

ANNEE     RECETTES     DEPENSES     BENEFICES
          ATTENDUES    ATTENDUES    ATTENDUS


   1        12000         6500         5500
   2        17000         7000        10000
   3        32000         9000        23000
   4        22000         9500        12500

BENEFICE ACTUALISE : -19965.29
TAUX DE RENTABILITE IMMEDIATE  9.17 %
L'INVESTISSEMENT PRØJETE N'EST PAS RENTABLE
```

```
CØUT INITIAL : 48000
DUREE DE VIE : 6
TAUX D'ACTUALISATIØN EN % : 9.5
RECETTES ATTENDUES : 33500,35000,38000,43000,39000,36000
VALEUR RESIDUELLE : 0
DEPENSES ATTENDUES : 20000,22000,24000,25000,29000,31000

ANNEE     RECETTES     DEPENSES     BENEFICES
          ATTENDUES    ATTENDUES    ATTENDUS


   1        33500        20000        13500
   2        35000        22000        13000
   3        38000        24000        14000
   4        43000        25000        18000
   5        39000        29000        10000
   6        36000        31000         5000

BENEFICE ACTUALISE :    7607.26
TAUX DE RENTABILITE IMMEDIATE 28.13 %
LE DELAI DE RECUPERATIØN EST DE  6 ANNEES
TAUX INTERNE DE RENTABILITE 15.20 %
```

12.2.8. Résolution d'un système linéaire.

L'organigramme n'est pas difficile à dessiner mais est assez long.

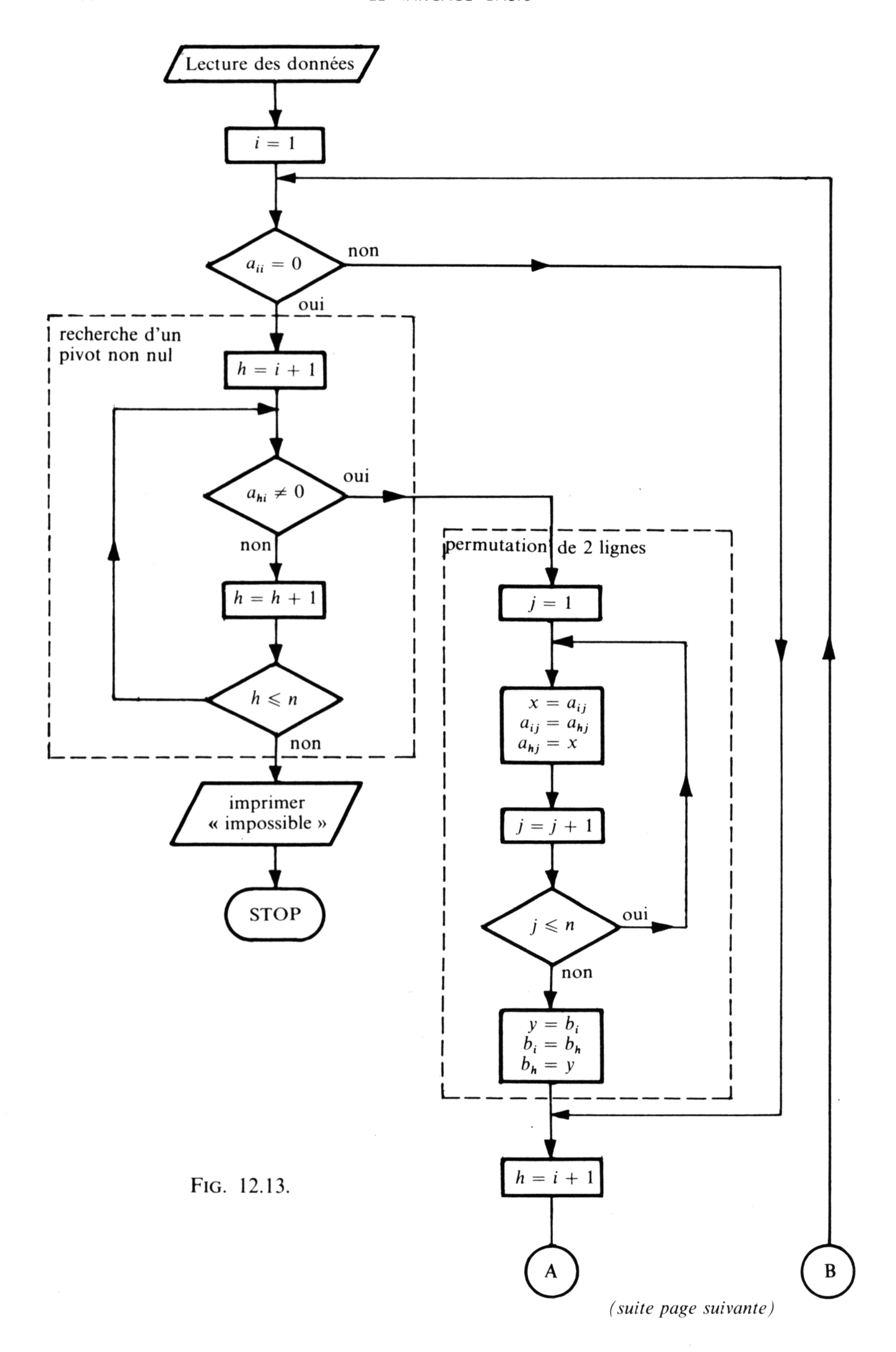


FIG. 12.13.

(suite page suivante)

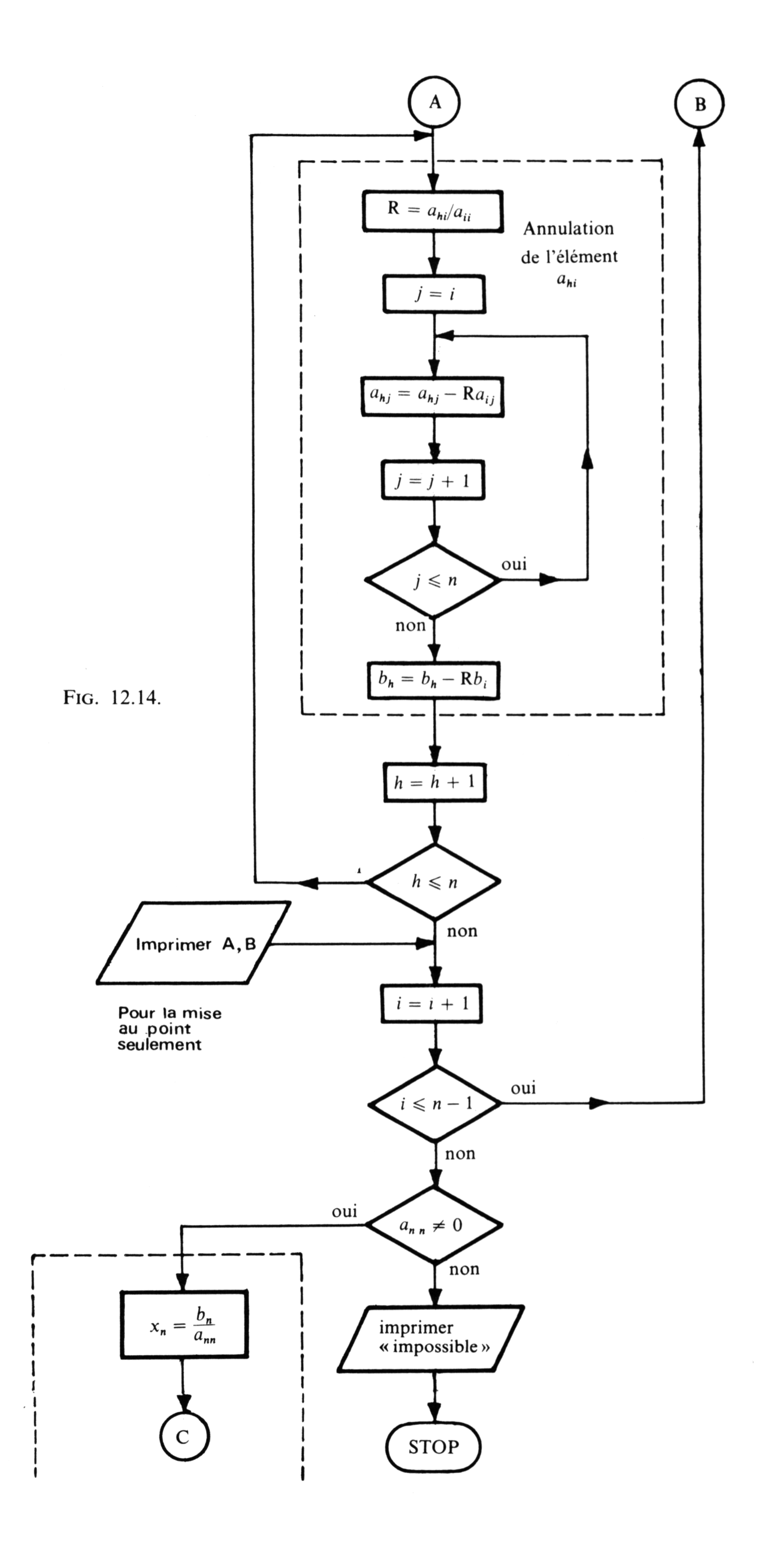

A
B
$R = a_{hi}/a_{ii}$
Annulation
de l'élément
a_{hi}
$j = i$
$a_{hj} = a_{hj} - Ra_{ij}$
$j = j + 1$
$j \leqslant n$
oui
non
$b_h = b_h - Rb_i$
$h = h + 1$
$h \leqslant n$
non
Imprimer A, B
Pour la mise
au point
seulement
$i = i + 1$
$i \leqslant n - 1$
oui
non
$a_{nn} \neq 0$
oui
non
$x_n = \frac{b_n}{a_{nn}}$
C
imprimer
« impossible »
STOP

FIG. 12.14.

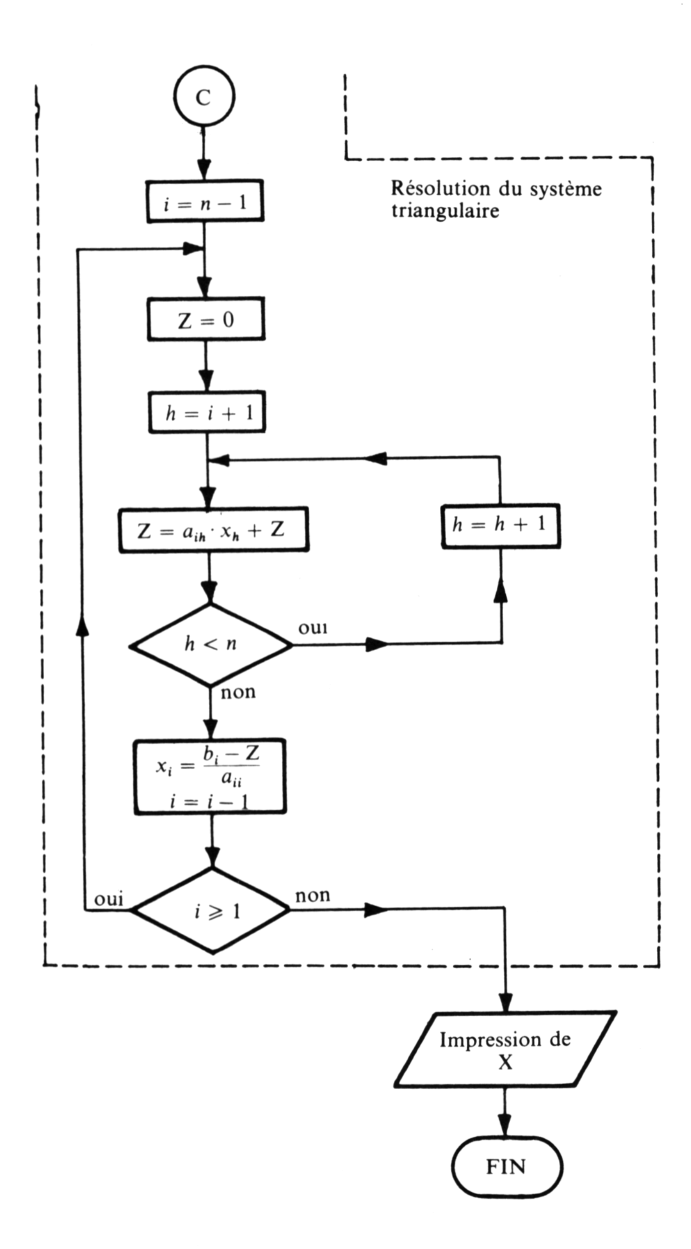

C
$i = n - 1$
Résolution du système triangulaire
$Z = 0$
$h = i + 1$
$Z = a_{ih} \cdot x_h + Z$
$h = h + 1$
$h < n$
oui
non
$x_i = \frac{b_i - Z}{a_{ii}}$
$i = i - 1$
oui
$i \geqslant 1$
non
Impression de X
FIN

FIG. 12.14 *bis*.

```
0010 DIM A(4,4),X(4),B(4)
0020 LET N=4
0030 MAT READ A,B
0035 MAT PRINT USING 215,A,B
0039 REM TRIANGULARISATIØN DU SYSTEME
0040 FØR I=1 TØ N-1
0049 REM TEST: PIVØT NØN NUL
0050   IF A(I,I)<>0 THEN 140
0055   REM RECHERCHE D'UN PIVØT NØN NUL
0060   FØR H=I+1 TØ N
0070     IF A(H,I)<>0 THEN 100
0080   NEXT H
0090   PRINT "IMPØSSIBLE"
0095  STØP
0097 REM PERMUTATIØN DE LIGNES
0100   FØR J=1 TØ N
0110     X=A(I,J)
0115     A(I,H)=A(H,J)
0117     A(H,J)=X
0120   NEXT J
0130   Y=B(I)
0135   B(I)=B(H)
0137   B(H)=Y
0140   FØR H=I+1 TØ N
0150     R=-A(H,I)/A(I,I)
0160     FØR J=I TØ N
0170       A(H,J)=A(H,J)+R*A(I,J)
0180     NEXT J
0190     B(H)=B(H)+R*B(I)
0200   NEXT H
0205 NEXT I
0210 PRINT
0215 :#####.##    #####.##    #####.##    #####.##
0220 MAT PRINT USING 215,A,B
0230 IF A(N,N)=0 THEN 90
0235 REM RESØLUTUIØN DU SYSTEME TRIANGULAIRE
0240 X(N)=B(N)/A(N,N)
0250 FØR I=N-1 TØ 1 STEP -1
0260   Z=0
0270   FØR H=I+1 TØ N
0280     Z=A(I,H)*X(H)+Z
0290   NEXT H
0300   X(I)=(B(I)-Z)/A(I,I)
0310 NEXT I
0320 PRINT "VECTEUR X"
0330 MAT PRINT USING 215,X
0340 END
5000 1,2,3,4
5005 7,3,5,1
5010 0,4,6,1
5015 1,9,9,3
5040 30, 32, 30, 58
```

} correspond à des DATA pour la plupart des systèmes (cf. cas particulier du paragraphe 5.3.1.)

```
 1.00         2.00         3.00         4.00

 7.00         3.00         5.00         1.00

 0.00         4.00         6.00         1.00

 1.00         9.00         9.00         3.00


30.00        32.00        30.00        58.00


 1.00         2.00         3.00         4.00

 0.00    -   11.00    -   16.00    -   27.00

 0.00         0.00         0.18    -    8.82

 0.00    -    0.00         0.00    -  221.00


30.00    -  178.00    -   34.73    -  884.00
VECTEUR X
 1.00         2.00         3.00         4.00


STØP AT LINE  0340
```

12.2.9. Minimisation de câblage selon l'algorithme de Kruskal.

Pour mettre en œuvre la méthode exposée en 12.1.9, nous allons utiliser les tableaux suivants :

Tableau A a deux dimensions : contient la matrice des coûts :

= coût de la liaison du nœud d'indice *i* au nœud d'indice *j*.

Tableau B et C a une dimension : contiendront les indices des nœuds à relier. La construction du domaine Ω consistera donc à remplir d'abord B(N) et C(N) avec les indices des nœuds dont la liaison réalise le coût minimum. Ensuite, nous remplirons B(N – 1) et C(N – 1), etc.

Dans l'exemple précédent, nous devrons obtenir en fin de traitement pour B et C les valeurs suivantes :

Au départ, B(I) et C(I) seront respectivement initialisés à 1 et 0. Un indice H permettra dans B de délimiter le domaine B(1) à B(H) qui contient les indices des nœuds qui ne font pas encore partie de Ω et B(H + 1) à B(N) les indices des nœuds qui font partie de Ω auquel il faut ajouter C(H + 1).

L'organigramme suivant correspond à l'une des solutions possibles.

I	B	C	
1		0	} inutile
2	2	4	
3	8	9	
4	6	8	
5	3	7	
6	1	3	
7	2	1	
8	6	2	
9	5	6	

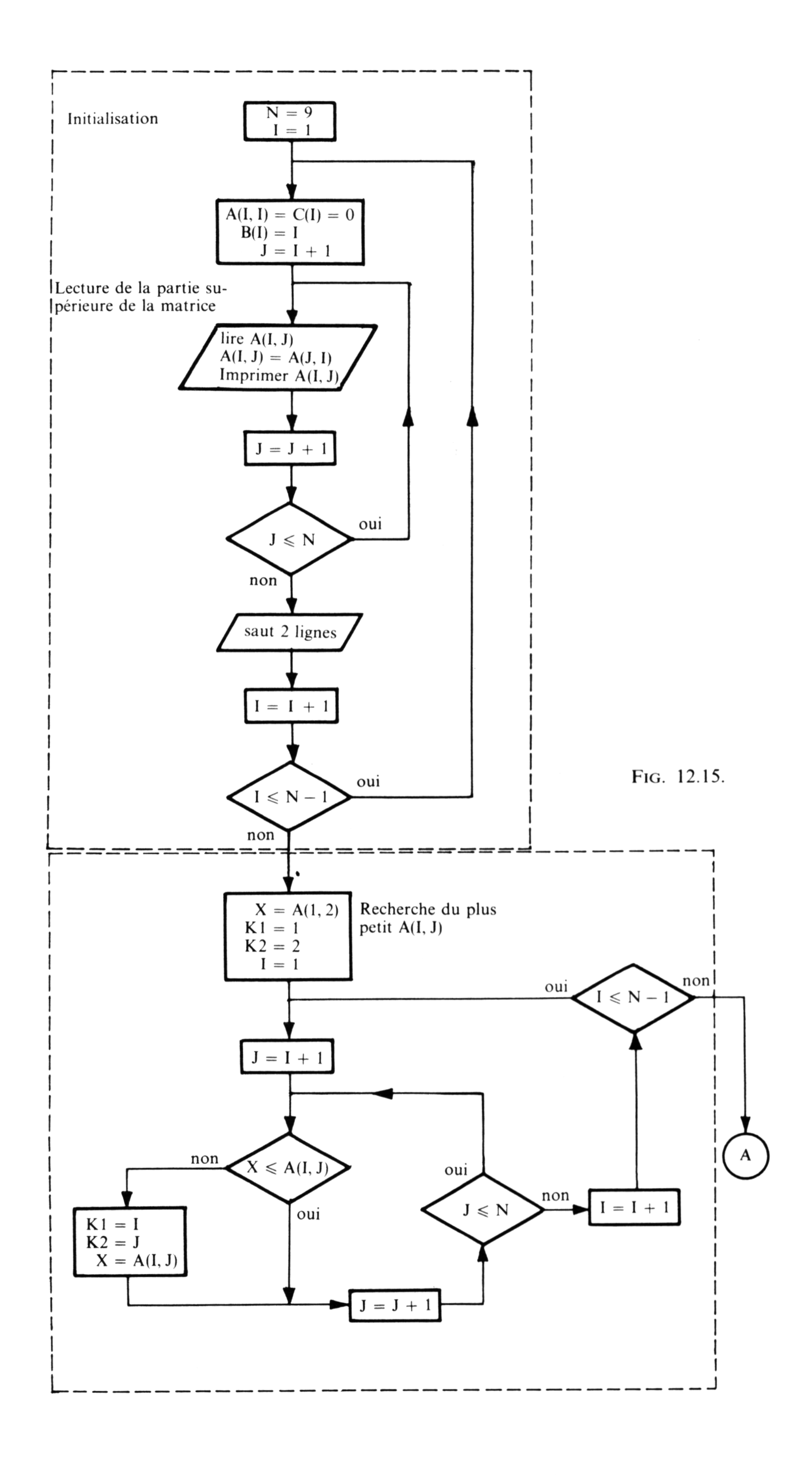

Initialisation
N = 9
I = 1
A(I, I) = C(I) = 0
B(I) = I
J = I + 1
Lecture de la partie supérieure de la matrice
lire A(I, J)
A(I, J) = A(J, I)
Imprimer A(I, J)
J = J + 1
J ⩽ N
oui
non
saut 2 lignes
I = I + 1
I ⩽ N – 1
oui
non
X = A(1, 2)
K1 = 1
K2 = 2
I = 1
Recherche du plus petit A(I, J)
oui
I ⩽ N – 1
non
J = I + 1
non
X ⩽ A(I, J)
oui
K1 = I
K2 = J
X = A(I, J)
J = J + 1
oui
J ⩽ N
non
I = I + 1
A

FIG. 12.15.

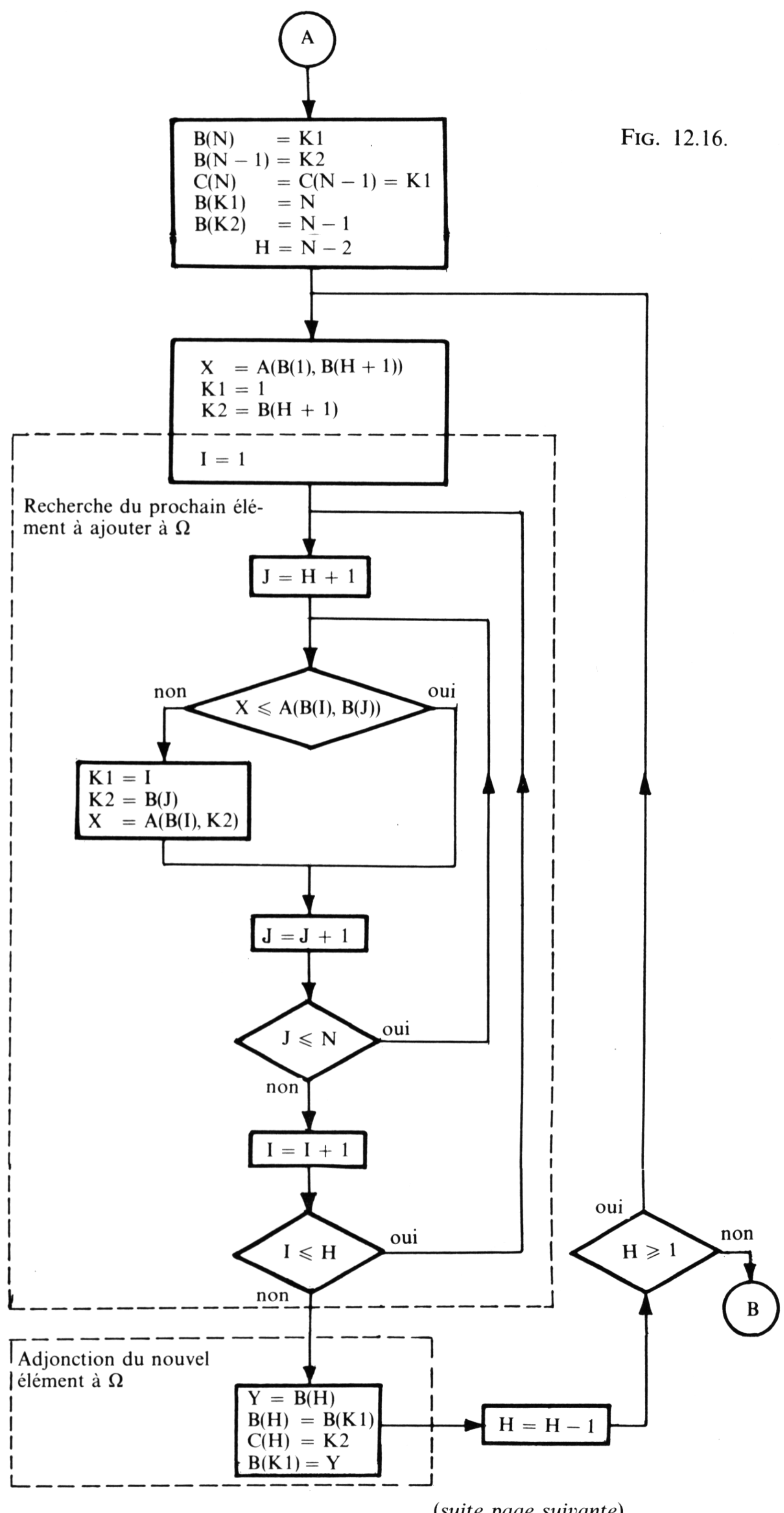


FIG. 12.16.

(*suite page suivante*)

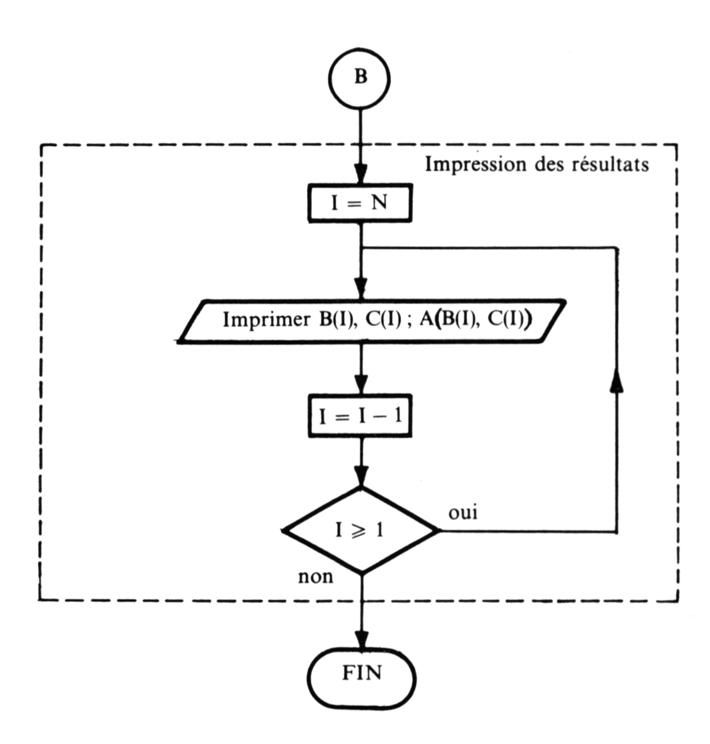

B
Impression des résultats
I = N
Imprimer B(I), C(I) ; A(B(I), C(I))
I = I − 1
I ⩾ 1
oui
non
FIN

FIG. 12.16 *bis*

```
0010 DIM A(9,9),B(9),C(9)
0020 LET N=9
0030 FØR I=1 TØ N-1
0035   LET A(I,I)=C(I)=0
0040   LET B(I)=I
0045   FØR J=I+1 TØ N
0050     READ A(I,J)
0055     LET A(J,I)=A(I,J)
0060     PRINT USING 65, A(I,J),
0065 :##.### ##.### ##.### ##.### ##.### ##.### ##.### ##.### ##.###
0070   NEXT J
0075   PRINT //
0080 NEXT I
0110 LET X=A(1,2)
0120 LET K1=1
0130 LET K2=2
0140 FØR I=1 TØ N-1
0150   FØR J=I+1 TØ N
0160     IF X<=A(I,J) THEN 200
0170     LET K1=I
0180     LET K2=J
0190     LET X=A(I,J)
0200   NEXT J
0210 NEXT I
0220 LET B(N)=K1
0230 LET B(N-1)=K2
0240 LET C(N)=C(N-1)=K1
0250 LET B(K1)=N
0260 LET B(K2)=N-1
0270 REM
0280 REM DEBUT DE LA BØUCLE SUR H
0290 REM
0300 FØR H=N-2 TØ 1 STEP -1
0310  LET X=A(B(1),B(H+1))
0315   LET K1=1
0317   LET K2=B(H+1)
0320   FØR I=1 TØ H
0330     FØR J=H+1 TØ N
0340       IF X<=A(B(I),B(J)) THEN 380
0350       LET K1=I
0360       LET K2=B(J)
0370       LET X=A(B(I),K2)
0380     NEXT J
0390   NEXT I
0400   LET Y=B(H)
0410   LET B(H)=B(K1)
0420   LET C(H)=K2
0430   LET B(K1)=Y
0440 NEXT H
0490 PRINT " I    J   A(I,J)"
0500 FØR I=N-1 TØ 1 STEP -1
0510   PRINT USING 515 ,B(I),C(I),A(B(I),C(I))
0515 :###  ###  ##.###
0520 NEXT I
0530 END
5000 2.95,2.1,6.2,3.4,4.2,4.7,7.5,8.1
5005 4.7,3.4,3.4,2.95,6.3,6.2,7.5,7.5
5010 3.4,4.7,2.95,7.5,7.5
5015 4.7,3.4,8.1,4.7,6.75
5020 1.5,3.4,4.2, 4.7
5025 4.7, 3.4, 4.45
5030 6.2, 5.4
5035 2.1
```

Lignes 0050 : lecture de la partie triangulaire supérieure de la matrice

Ligne 0055 : compléter la matrice par symétrie

Ligne 0060 : impression de la partie supérieure de la matrice

Lignes 0130 à 0210 : recherche du plus petit A(I, J) et des indices I et J correspondants qui seront rangés dans K1 et K2

Lignes 5000 à 5035 : ceci correspond à des DATA pour la plupart des systèmes

```
2.950   2.100   6.200   3.400   4.200   4.700   7.500   8.100

4.700   3.400   3.400   2.950   6.300   6.200   7.500

7.500   3.400   4.700   2.950   7.500   7.500

4.700   3.400   8.100   4.700   6.750

1.500   3.400   4.200   4.700

4.700   3.400   4.450

6.200   5.400

2.100

 I      J    A(I,J)
 5      5    0.000
 6      5    1.500
 2      6    2.950
 1      2    2.950
 3      1    2.100
 7      3    2.950
 8      6    3.400
 9      8    2.100
 4      2    3.400
```

Manuels des constructeurs

Systèmes	*Sociétés*	*Titres*
XDS-940	Cegos-Tymshare	Manuel d'utilisation Super basic Super basic reference series supplement
	Télésystèmes	New basic reference manual (ref. 98 A 9952 031)
CDC 6000	CDC	Introduction to expanded basic
	SIA	Manuel d'utilisation Xbasic
Call 360	IBM	Call 360 Basic manuel de reference introduction au basic
XDS Sigma 5/7 (CII 10070)	CII RXDS	10070 basic (ref. 90154613/En)
HB 600 HB 6000	Honeywell-Bull	Basic language MARK II (ref. 711224c de General Electric)
Pdp 11	DEC	RSTS 11 Basic plus user's guide PL 11-71-01-01-A-D
Nova	Data General	Extended Basic user's manual (ref 093-000065-01)
Multi-8	Intertechnique	Basic Multi-8 Manuel de référence
HP 2100	Hewlett-Packard	A guide to time shared Basic (HP 02000-90016)
HP 9845	Hewlett-Packard	Graphics programming Techniques
TS9200	Philips	Time sharing BASIC (ref. 512291123 071)
Varian 620	Varian	Varian 620 Basic language reference manual (ref. 98 A9952031)

Pour alléger le texte (en particulier dans le chapitre 11), les extensions communes aux systèmes CIGI-TYMSHARE et TÉLÉSYSTÈMES ou les extensions spécifiques à l'un de ces systèmes sont regroupées sous le nom de système XDS 940.

INDEX ALPHABÉTIQUE

Photocomposé et réimprimé en décembre 1979
par Joseph FLOCH, Maître Imprimeur à Mayenne
N° 6952

Dépôt légal : 4e trimestre 1979
N° d'Éditeur : 3540

ÉDITIONS EYROLLES

8563


EYROLLES
LE LANGAGE BASIC
ET SES EXTENSIONS
J.-P. LAMOITIER



Document numérisé avec amour par :

AMSTRAD CPC

MÉMOIRE ÉCRITE

ACME

https://acpc.me/